MCSDK UG Chapter Exploring

MCSDK User Guide: Exploring the MCSDK

Last updated: **01/06/2017**

Contents

Overview

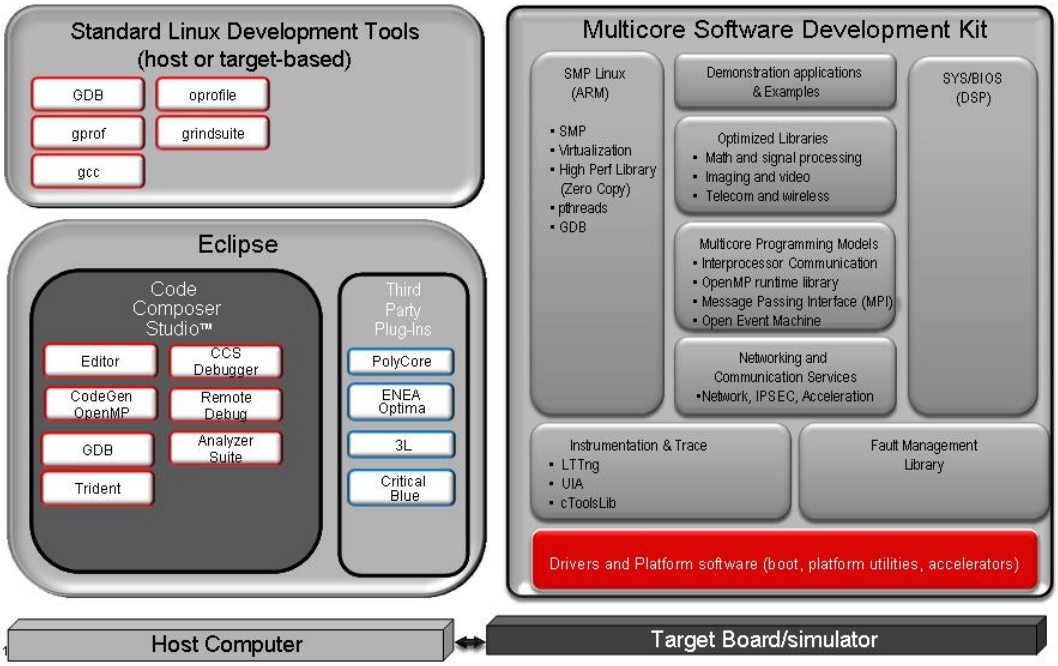
The Multicore Software Development Kit (MCSDK) provides foundational software for TI KeyStone II device platforms. It encapsulates a collection of software elements and tools intended to enable customer application development and migration.

The foundational components include:

- SYS/BIOS real-time embedded operating system on DSP cores
- Linux high-level operating system running on ARM A15 cluster (SMP mode)
- DSP chip support libraries, DSP/ARM drivers, and basic platform utilities
- Interprocessor communication for communication across cores and devices
- SoC resource management
- Optimized application-specific (transport) and application non-specific algorithm libraries
- Trace debug and instrumentation
- Bootloaders and boot utilities, power-on self test
- Demonstrations and examples
- ARM software libraries available in Linux devkit or via Arago/Yocto
- Latest toolchain (ARM Linaro, DSP TI CodeGen)
- Host tools, integrated development environment

This Chapter provides a high level overview of the different pieces so you will gain a sense of what they are and what they do. After reading this chapter you should have a sense of the different pieces that make up the MCSDK. You can then refer to the chapter, Developing with the MCSDK to get the details.

Here is a high-level picture of the software ecosystem that MCSDK is a part of:



Note: Not all items in the genric picture above applies to all parts.

Please see the release notes for the comprehensive list of components and version; the latest release notes is MCSDK Release Notes (http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/exports/MCSDK_Release_Notes.pdf). For licensing information, please see the software manifest; the latest software manifest is MCSDK Software Manifest (http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/exports/MCSDK_Software_Manifest.pdf).

IS NOT

- Support for BIOS5 or older releases
- Support for CCS 4.x or older releases
- Support for platforms not listed here (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Getting_Started#Supported_Devices_and_Platforms)
- DSP image format other than ELF (e.g., COFF)
- Big endian ARM
- Big endian DSP with ARM Little endian configuration

Acronyms

Put a condensed acronym table here. That is one that addresses acronyms that this Chapter uses. This means you should add/remove from the Table below.

The following acronyms are used throughout this chapter.

Acronym	Meaning
AMC	Advanced Mezzanine Card
CCS	Texas Instruments Code Composer Studio
CSL	Texas Instruments Chip Support Library
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol
DSP	Digital Signal Processor
DVT	Texas Instruments Data Analysis and Visualization Technology
EDMA	Enhanced Direct Memory Access
EEPROM	Electrically Erasable Programmable Read-Only Memory
EVM	Evaluation Module, hardware platform containing the Texas Instruments DSP
HUA	High Performance Digital Signal Processor Utility Application
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IPC	Texas Instruments Inter-Processor Communication Development Kit
JTAG	Joint Test Action Group
MCSA	Texas Instruments Multi-Core System Analyzer
MCSDK	Texas Instruments Multi-Core Software Development Kit
NDK	Texas Instruments Network Development Kit (IP Stack)
NIMU	Network Interface Management Unit
PDK	Texas Instruments Programmers Development Kit
RAM	Random Access Memory

RTSC	Eclipse Real-Time Software Components
SPL	Secondary Program Loader
SRIO	Serial Rapid IO
TCP	Transmission Control Protocol
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UIA	Texas Instruments Unified Instrumentation Architecture
USB	Universal Serial Bus

Note: We use the abbreviation TMS when referring to a specific TI device (processor) and the abbreviation TMD when referring to a specific platform that the processor is on. For example, TMS320C6678 refers to the C6678 DSP processor and TMDSEVM6678L refers to the actual hardware EVM that the processor is on.

ARM Subsystem

Overview

ARM subsystem runs following software components:-

- U-Boot - Boot loader
- Boot Monitor - Monitor and other secure functions
- SMP Linux - ARM A15 port of SMP Linux

This section describes details of these components delivered as part the Linux ecosystem in MCSDK.

Linux

This section provides information on the features, functions, delivery package and compile tools for the Linux Kernel release for KeyStone II. This document describes how to install and work with Texas Instruments' Linux Kernel for the KeyStone II platform. The MCSDK provides a fundamental Linux based software platform for development, deployment and execution on the ARM A15 processor of the KeyStone II architecture. In this context, the document contains instructions to:

- Install the release
- Build the sources contained in the release and run it on the target keystone II EVM
- Kernel and peripheral driver design and/or implementation details

Changes impacting users

It is strongly recommended to upgrade all images to the same release which includes: uboot, boot monitor, kernel and file system.

This section describes the list of changes that impacts the user

K2_RT_LINUX_03.10.10_13.11/K2_LINUX_03.10.10_13.11

Starting with this release, the git repo for linux, boot monitor and u-boot has been moved to git.ti.com (previously on Arago). Also Linux kernel is rebased to v3.10.10. Please see the build location for the URL for these repos.

K2_RT_LINUX_03.08.04_13.09_01/K2_LINUX_03.08.04_13.09_01

In these tags, kernel started using 512M capacity of the NAND available on the EVM. The ubifs partion size in dts file is changed accordingly to use the increased size. The u-boot is upgraded to increase the mtd partition size to reflect the new size. The u-boot tag for this is K2_UBOOT_2013-01_13.09_01. There are dts changes that requires the kernel to be in sync with DTS.

DEV.MCSDK-03.08.04.11

- DTS file name is changed to match with EVM name, k2hk-evm.dts. So the DTB file name has k2hk to indicate the EVM name.

DEV.MCSDK-03.08.04.10

- Linux upstream kernel changed to 3.8.4. Last release was based on 3.6.6. As a side effect, the location of dtb file generated by the Linux kernel build is now at arch/arm/boot/dts folder.
- Changed the name of dtb file populated in the release to uImage-tci6638-evm.dtb.

DEV.MCSDK-03.06.06.08

- Changed the name of u-boot command install_skern() to mon_install()
- Changed the name of dtb file populated in the release binaries from tci6638-evm.dtb to uImage-keystone-evm.dtb

Installation Guide

Prerequisites for MCSDK 3.0 Installation

Before you begin with the installation of this package please make sure you have met the following system requirements:

- CCS v5.3.0 or later: Available on Windows or Linux Host
- Keystone II Simulator: Available on Windows or Linux Host
- Toolchain: **Must** be installed on Linux Host. Ubuntu 12.04 recommended.

- Terminal Emulator: Tera Term on Windows or Minicom on Linux Host can be used.

For CCS and Simulator installation

please refer to corresponding documentation, which comes with CCS and/or Simulator. Please see CCS Getting started guide (http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide)

Toolchain Installation

Please refer to Linaro toolchain (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Tools#Linaro_toolchain)

Linux Software Installation

The Linux kernel related utilities and pre-built binaries are provided in the release under the folder mcsdk_linux_<version>.

Please refer the "Getting Started" section for more details on installation.

Tag and Branch naming convention

U-Boot

U-Boot release tag is named in the following format:-

- K2_UBOOT_<upstream release version><year_> <month_><[iteration]>

For Example, K2_UBOOT_2013-01_13.08 indicates, the u-boot is based on upstream version 2013-01 and the release is done in August 2013. This done to make it intuitive for anyone to identify the baseline upstream u-boot version used in a release from the tag name.

Linux

Linux release tag is named in the following format:-

- K2_LINUX_-<upstream release version_><year_> <month_><[iteration]>

For example K2_LINUX_03.08.04_13.08_01 indictaes the Linux kernel is based on upstream kernel version 03.08.04 and the release is done in August 2013 and iteration is 1. Iteration is optional and is used when multiple tags are to be used for the same month for what ever reason.

For RT Preempt patched kernel, the tag name also include the word "RT"

RT Preempt patched Linux kernel is available on a seperate master branch (master-rt). master branch is without RT patches. This will be available under release_<build number>/master-rt and release_<build number>/master branches respectively on Arago git repo.

Boot Monitor

Boot Monitor release tag is named in the following format:-

- K2_BM_<year_> <month_><[iteration]>

For example K2_BM_13.08 is released in August 2013

Build Instructions

There are two possible ways to build the linux kernel:

- The first is to build the standalone kernel by cloning the kernel Yocto git repository on the local host. Please refer the Yocto section below for details.
- The second is to build the kernel together with the filesystem using Yocto project on Arago

Build Prerequisites

- Ubuntu 12.04 LTS distribution and sudo access should be available to the user.
- Install the tool chain as described in the section **Toolchain Installation** above for cross compiling.
- Install and configure git in the Ubuntu machine. Use the following command to install git:

```
$ apt-get install git-core
```

- To configure git please refer here[1] (http://kernel.org/pub/software/scm/git/docs/git-config.html). If your network is behind a proxy, those settings need to be configured as well.
- Packages needed at build-time can be fetched with a simple command on Ubuntu 12.04:

```
$ sudo apt-get install build-essential subversion ccache sed wget cvs coreutils unzip texinfo docbook-utils gawk help2man diffstat file g++ texi2html bison flex htmldoc chrpath
libxext-dev xserver-xorg-dev doxygen bitbake uboot-mkimage libncurses5-dev
```

Note: If you are running a distribution other than Ubuntu 12.04 LTS, please refer to your distribution documentation for instructions on installing these required packages.

Note: From the MCSDK 3.0.4 release onward, the size of the the rootfs has increased beyond 80MB. As a result use of ramfs is not possible. The filesystem needs to be under 80M for use with ramfs

Proxy Setup

If your network is behind a firewall/proxy additional settings are needed for bitbake to be able to download source code repositories for various open source projects. Some of these configuration items are:

- wgetrc: A ".wgetrc" needs to be created under the $HOME directory. A sample wgetrc can be found here[2] (http://www.gnu.org/software/wget/manual/html_node/Sample-Wgetrc.html). Please update configuration variables http_proxy, https_proxy and ftp_proxy as per your network environment.
- Set proxy environment variables. These may be added to your .bashrc or shell initialization script:

```
export http_proxy="http://<your_proxy>:<port>"
export ftp_proxy="http://<your_proxy>:<port>"
export https_proxy="http://<your_proxy>:<port>"
```

- $HOME/.subversion/servers needs to be updated if the network is behind a proxy. The following lines need to be modified as per settings for your network:

```
http-proxy-exceptions = "exceptions"
http-proxy-host = "proxy-host-for-your-network"
http-proxy-port = 80
```

U-Boot Build instructions

First clone the U-Boot source tree from Arago git repository

```
$ git clone git://git.ti.com/keystone-linux/u-boot.git u-boot-keystone
$ cd u-boot-keystone
$ git reset --hard <Release tag>
where release tag can be obtained from the release notes. For example, release tag used is DEV.MCSDK-2013-01.11
```

1. To build u-boot.bin that can be loaded and run from MSMC SRAM, do the following

```
make <soc>_evm_config
make
```

To do 2 stage SPI NOR boot, following images are to be built.

2. To build u-boot-spl.bin that can be booted from SPI NOR flash, do the following:-

```
make <soc>_evm_config
make spl/u-boot-spl.bin
```

The u-boot-spl.bin will be available at spl/ folder

3. To build secondary boot u-boot.img (in uImage firmware format) that can be flashed and booted through SPI NOR flash do the following:-

```
make <soc>_evm_config
make u-boot.img
```

The u-boot.img will be created at root directory of u-boot source tree.

Alternately both u-boot-spl.bin and u-boot.img can be combined and flashed to SPI NOR flash. To do so, use the following command to build a single gph image for programming on SPI NOR flash

```
make <soc>_evm_config
make u-boot-spi.gph
```

The u-boot-spi.gph image will be created in the root folder.

4. To build a single u-boot-nand.gph image that can be programmed on EMIF16 NAND flash, do the following:-

```
make <soc>_evm_config
make u-boot-nand.gph
```

Note: <soc> is "k2hk", "k2l" or "k2e"; for MCSDK 3.0.x, <soc> is "tci6638"

Boot Monitor Build instructions

To build boot monitor code, first clone the git repository as

```
$ git clone git://git.ti.com/keystone-linux/boot-monitor.git
$ cd boot-monitor
$ git reset --hard <Release tag>
where release tag can be obtained from the release notes. For example, DEV.MCSDK-03.00.00.11 is the release tag.

$ make clean
$ make
```

skern-<soc>.bin (for MCSDK 3.0.x, skern.bin) will be created in the current working directory.

Note: <SOC> is "k2hk", "k2l" or "k2e"

Linux kernel and the device tree blob build instructions

This section assumes that the Linaro toolchain for ARM is installed and environment variables CROSS_COMPILE, ARCH are set up as per instructions given in section Toolchain Installation Linaro_toolchain (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Tools#Linaro_toolchain).
(e.g)
export CROSS_COMPILE=arm-linux-gnueabihf-
export ARCH=arm
PATH=<path to installed toolchain>/bin:$PATH

The first step in building the kernel is downloading the kernel source code from the git repositories to the local Ubuntu 12.04 host. This is performed as follows:

```
$ git clone git://git.ti.com/keystone-linux/linux.git linux-keystone
$ cd linux-keystone
$ git reset --hard <Release tag> where release tag can be obtained from Release notes. For example tag is DEV.MCSDK-03.08.04.11 for the release.
```

Before building the linux kernel for Simulator, user has to modify the source file linux-keystone/drivers/net/ethernet/ti/keystone_ethss.c

The line u8 mac_addr_nic[6] = {0x5c, 0x26, 0x0a, 0x80, 0x0d, 0x43}; /* NIC addr */ has to be modified

The NIC address should be appropriately modified with the NIC address of the host where CCS and the simulator are installed.

To build the kernel for execute the following commands inside the linux-keystone folder:

```
$ make keystone2_defconfig
$ make uImage
```

The next step is to build the device tree file:

For building DTB for EVM use the following command:-

```
$ make <soc>-evm.dtb
```

Note: <soc> is "k2hk", "k2l" or "k2e"

For building DTB for simulator use the following command:-

```
$ make keystone-sim.dtb
```

Note that starting 3.8.4, location of dtb binary is moved to arch/arm/boot/dts.

To Build Linux for Full RT Preempt mode, do the following

```
$ git reset --hard <Release tag> where release tag can be obtained from Release notes. For example tag for RT kernel is DEV.MCSDK-RT-03.08.04.11 for the release.
$ make keystone2_fullrt_defconfig
$ make uImage
```

TransportNetLib Software Library and Build instructions

TransportNetLib software package provides a library for ARM user space application to gain direct access to networking h/w. This is for such use cases as fast path packet processing. The TransportNetLib includes HighPerformanceLib(HPLIB) and Network API(NetAPI) modules. Detailed description of these modules and build insructions are available at http://processors.wiki.ti.com/index.php/TransportNetLib_UsersGuide.

Running U-Boot, Boot Monitor and Linux Kernel on EVM

Loading and Running U-Boot on EVM through UART

Before loading the u-boot to EVM you need to create u-boot.uart blob. That is the u-boot.bin binary with preceding 4096 zeros. Assuming you are in the u-boot source top level directory execute the following commands after building u-boot:

```
> dd if=/dev/zero of=4k_zeros bs=4096 count=1
> cat 4k_zeros u-boot.bin > u-boot.uart
```

You should run two terminals on a PC. One connected to the BMC and another to the UART0 of the K2HK SOC. The terminal connected the the SOC UART0 port must support xmodem protocol. For example to use minicom on Linux, follow the steps at [[3] (http://processors.wiki.ti.com/index.php/Setting_up_Minicom_in_Ubuntu)]

At the UART0 terminal, user should initiate transfer of the u-boot.uart file using the xmodem protocol. User should see

```
"s: Give your local XMODEM receive command now."
```

Using BMC console set the ARM UART boot mode and reboot the K2HK SOC.

```
BMC> bootmode #4
BMC> reboot
```

This would initiate the transfer. Once transfer is complete, user would see

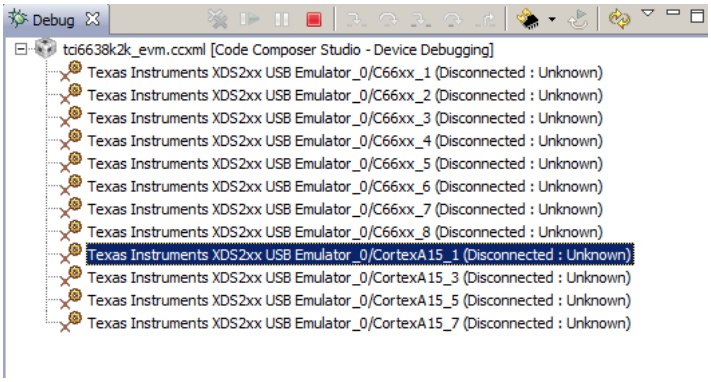
```
Transfer complete
READY: press any key to continue...
```

Once any key is pressed, u-boot starts up and boot up log is seen at the UART0 terminal

Loading and Running U-Boot on EVM through CCS

If you do not have a target configuration for the EVM, you have to create one. Create a new target configuration for the TCI6638 device using the "Texas Instruments XDS2xx USB Emulator" connection. You have to do that only once.

Launch the target configuration.



Power on the EVM and connect CCS to the CortexA15_1 target.

Connect serial cable between the PC and the EVM. Open Teraterm or Hyper Terminal, create a connection with 115200 baud rate, 8 data bits, no parity, 1 stop bits and no flow control.

Copy the images from the images folder of the installed release directory to the loadlin folder. Copy the latest tci6638-evm.ccxml to the loadlin folder. Also the script may need tweaking to suite the environment on which you are running this script. The example given here is for reference only.

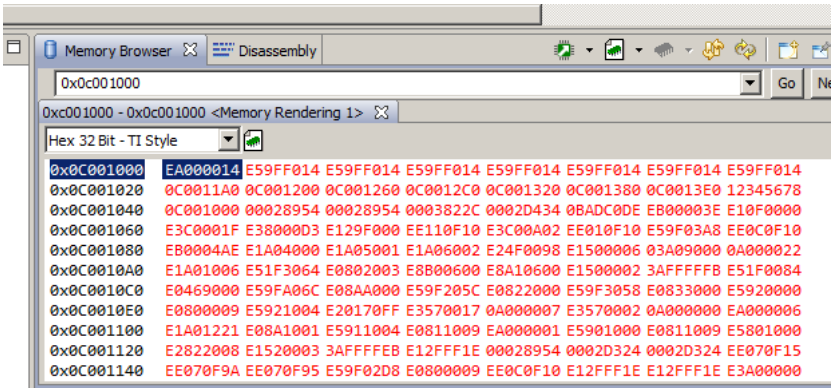
Edit the loadlin-evm-uboot.js java script from the <release folder>/host-tools/loadlin folder for the following

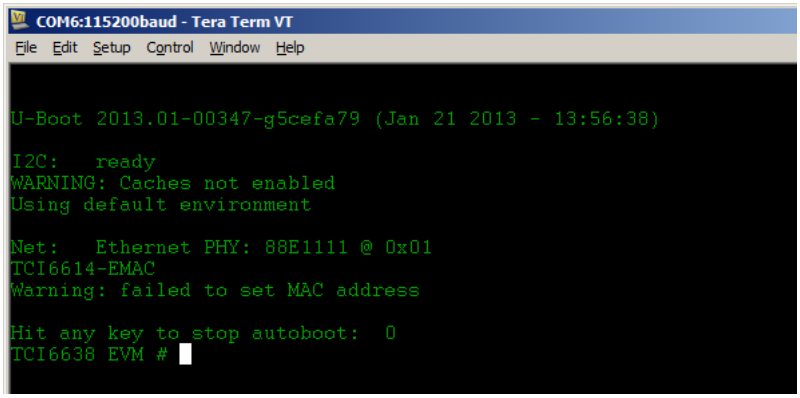
```
PATH_LOADLIN = <path of release folder>/host-tools/loadlin
var pathUboot = PATH_LOADLIN + "/u-boot-keystone-evm.bin";
var pathSKern = PATH_LOADLIN + "/skern-keystone-evm.bin";
```

Save the file. Copy the image files (u-boot-keystone-evm.bin and skern-keystone-evm.bin) from <release folder>/images to <release folder>/host-tools/loadlin. Open the scripting console and type

- loadJSFile "<path of release folder>/host-tools/loadlin/loadlin-evm-uboot.js"

This will load, u-boot image to MSMC RAM at 0xc001000 and boot monitor image to MSMC RAM at address 0xc5f0000. Make sure PC is currently pointing to 0xc001000. Click Resume button on the CCS window to run u-boot.





Loading and running U-Boot on EVM through SPI Boot

Two Stage SPI Flash Boot

The two stage SPI flash u-boot image consists of a first stage u-boot SPL binary with a header and a footer, and a second stage full u-boot binary. These two binaries can be programmed to SPI flash in two ways.

- Using a combined single GPH image (u-boot-spi-<soc>-evm.gph from release) to program the SPI NOR flash.

Where <soc> is k2hk/k2e/k2l. If you build u-boot from source code, u-boot-spi.gph will be created in the root source directory and is to be used as u-boot-spi-<soc>.gph. This is the quick and easiest way to upgrade u-boot image if you have u-boot running already on the evm. This implemented using a u-boot env script which is added to very recent version of the u-boot. To see if current u-boot on EVM support this, do

```
>printenv
```

and check if get_uboot_net and burn_uboot env variables are present. Also name_uboot is set to u-boot-spi-<soc>-evm.gph. If so do following to upgrade u-boot assuming gph image is copied to root directory of tftp server

```
env default -f -a
setenv serverip <ip address of tftp server>
setenv tftp_root <tftp root directory>
run get_uboot_net
run burn_uboot
reset
```

The EVM now boots with latest version of u-boot image.

'**NOTE:**' User might want to check the value of mem_reserve env variable to ensure only reserved memory is used by applications as described in Reserve DDR memory

- Program the two stage binaries separately, follow the steps in the below section

Burning First Stage u-boot-spl.bin to SPI Flash

Follow the same procedure described in section Burning Single Stage U-Boot to SPI Flash, but use the u-boot-spl.bin image instead. The SPL U-Boot reside in SPI flash at offset 0.

Burning Second Stage u-boot.img to SPI Flash

First step is to load u-boot.img to MSMC SRAM address 0xc300000 through CCS or other means. Then do the following steps:-

- sf probe
- sf erase 0x10000 <size of image in hex rounded to sector boundary of 0x10000>

  for example if the size of image is hex 0x4500b, round it to 0x50000
- sf write 0xc300000 0x10000 <size of image in hex>

Single Stage SPI Flash Boot

In the single stage SPI flash boot, the image to be burnt into the SPI flash contains the u-boot.bin with a header and a footer added.

Burning Single Stage U-Boot to SPI Flash

This section assumes that u-boot is booted up and running on EVM, e.g. using CCS (see section above)

Load u-boot.bin to memory to be saved to SPI flash through CCS

```
* Have a copy of u-boot.bin ready in a directory on host PC running CCS
* Start and connect CCS to ARM core 0
* CCS > Run > "Suspend"
* CCS > Tools > Load Memory > browse to the copy of u-boot.bin and load it to address 0x0c300000
* CCS > Run > "Resume"
where 0x0c300000 is an example address to temporarily keep the u-boot.bin image.
```

Now a copy of u-boot.bin is loaded to 0x0c300000.

At the u-boot prompt:

- Format the u-boot.bin image loaded in memory to SPI boot format

```
> fmtimg spi 0x0c001000 0x0c300000 <hex_size>
where hex_size is the size of the u-boot.bin in hex bytes,
and 0x0c001000 is the address that u-boot.bin will be loaded by RBL and starts execution.
Note down the formatted image hex size printed out at the end of the command, call this fmt_img_size.
```

- Alternatively issue following commands to perform the formatting

```
> setenv fileaddr 0x0c300000
> setenv filesize <hex_size>
> fmtimg spi 0x0c001000
where command fmtimg takes the hex size of u-boot.bin from env variable "filesize",
and the address where u-boot.bin is situated is taken from env variable "fileaddr".
This is useful if u-boot.bin is loaded to memory by using tftp. In that case, filesize and fileaddr are set in the process.
Note down the formatted image hex size printed out at the end of the command, call this fmt_img_size.
```

- Prepare SPI flash for saving formatted u-boot.bin later

```
> sf probe
> sf erase 0 <hex_len>
erases hex_len bytes in SPI flash starting from offset 0,
where hex_len must be at least the formatted image size shown in the fmtimg command and on flash sector boundary,
sector size of SPI flash on EVM is 0x10000
e.g. if formatted image size is 0x2FBA0, then "sf erase 0 0x30000"
```

- Save formatted u-boot.bin image to SPI flash

```
> sf write 0x0c300000 0 <fmt_img_size>
where fmt_img_size is the hex size of the formatted image printed out in the fmtimg command above.
```

Booting U-Boot from SPI Flash on EVM

```
* Power off EVM
* Set Boot Setting DIP Switch (SW1) on EVM to 0010 (ARM SPI boot mode)
    - i.e SW1: 1(OFF) 2(OFF) 3(ON) 4(OFF)
* If any, disconnect CCS from EVM
* Power up EVM
```

Loading and running U-Boot on EVM through NAND boot (Keystone2 Rev2.0 or Later EVM ONLY)

Burning U-Boot to NAND Flash

This section assumes that u-boot is booted up and running on EVM, e.g. using CCS or SPI boot. Also the EVM is connected to a network where a tftp server is available.

- compile the nand u-boot image on host (refer to U-Boot Build instructions section)
- place u-boot-nand.gph in tftp server
- under u-boot prompt, do

```
> setenv serverip <tftp-server-ip-address>
> setenv bootfile '/tftp/server/path/to/u-boot-nand.gph'
> dhcp ${fileaddr}
> nand ecclayout set 1
> nand erase.part bootloader
> # For K2HK EVM
> nand write ${fileaddr} bootloader ${filesize}
> # For K2E or K2L EVM
> nand write ${fileaddr} <offset> ${filesize}
> nand ecclayout set 0
```

'**NOTE:**' There is an errata for K2L and K2E PG 1.0 that NAND boot does not distinguish between a block that has been pre-marked as bad vs one that is declared bad due to error correction failure. When a bad block is found, the ROM bootloader moves to the next block and re-initializes the boot data processor. When the block is pre-marked as bad the boot data processor should not be reset.

The workaround to this problem is to burn a multi-block image in the NAND with contiguous good blocks. Users can check the bad block list using "nand bad" command and skip any bad blocks.

The NAND device on K2L EVM has a block size of 256KB, and u-boot-nand.gph is over 300KB which will uses 2 blocks, starting from block 0 (offset address 0). Similarly the nand device on K2E EVM has a block size of 128KB, and u-boot-nand.gph will uses 3 blocks. If any of the 2 or 3 blocks are marked bad, we should skip the bad block and use the next 2 or 3 good blocks.

E.g. block 0 is good but block 1 is bad, and block 2 - 4 are good, the <offset> should be set to 0x80000 (starting offset address of block 2) for K2L EVM, and 0x40000 for K2E EVM.

'**NOTE:**' User might want to check the value of mem_reserve env variable to ensure only reserved memory is used by applications as described in Reserve DDR memory

Booting U-Boot from NAND Flash on EVM

To boot u-boot from NAND flash, do one of the following:

- By setting boot pin:

```
* Power off EVM
* Set Boot Setting DIP Switch (SW1) on EVM to 0000 (ARM NAND boot mode)
    - i.e SW1: 1(OFF) 2(OFF) 3(OFF) 4(OFF)
* Power up EVM
```

- From BMC terminal on Rev2.0 EVM

```
BMC> bootmode #0
BMC> reboot
```

Loading and Running Linux Kernel on EVM

Loading and Running Linux Kernel on EVM using CCS

First step is to load **u-boot** (eg. u-boot-keystone-evm.bin) and **boot monitor** (eg. skern-keystone-evm.bin) images onto MSMC RAM and run u-boot to get the u-boot prompt. See the instructions in section Loading and Running U-boot on EVM using CCS.

Now follow the steps below to load and run Linux through CCS.

Copy the images from the images folder of the release directory to the loadlin folder.

Edit the loadlin-evm-kern.js file and make sure the following variables points to valid paths for the images.

```
PATH_LOADLIN = <path of lsp-release-folder>/host-tools/loadlin
var pathKernel          = PATH_LOADLIN + "/uImage-keystone-evm.bin";
var pathDtb             = PATH_LOADLIN + "/uImage-k2hk-evm.dtb";
var pathInitrd           = PATH_LOADLIN + "/<name of min root file system, eg. tisdk-rootfs.cpio.gz>".
If the root fs image size is greater than 9M, make sure the bootargs in u-boot is set correctly to reflect the size.
Currently it defaults to 9M.
```

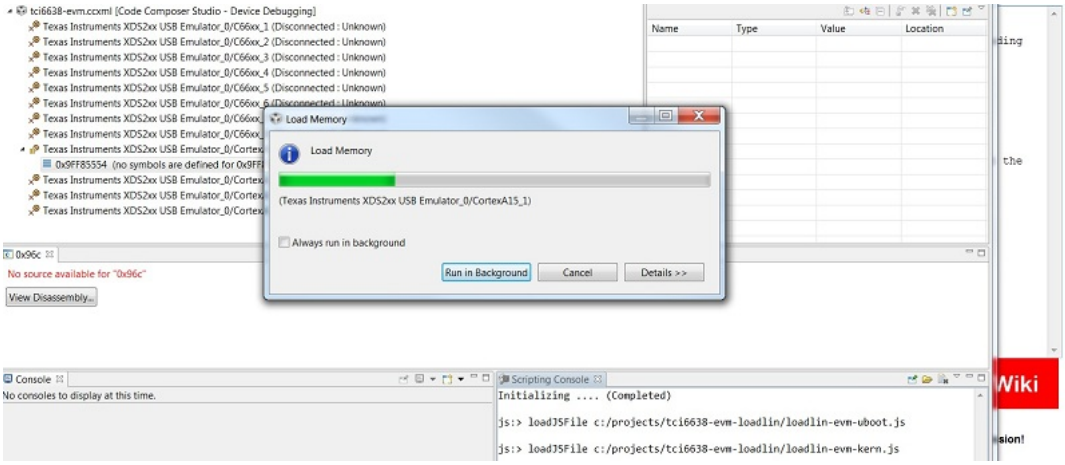
Note For RT Preempt images, use image file names uImage-rt-keystone-evm.bin and uImage-rt-k2hk-evm.dtb for kernel and dtb files.

Save the file and exit.

- Click onto the suspend button or Alt+F8

Open scripting console and run

- loadJSFile "<path of loadlin from release folder>/loadlin-evm-kern.js"



- The script loads Linux kernel image, DTB and file system image to DDR. Once script execution is complete, click onto the Resume button on CCS.
- At u-boot prompt, update the env variable bootargs, for example:

```
> setenv bootargs 'console=ttyS0,115200n8 rootwait=1 earlyprintk rdinit=/sbin/init rw root=/dev/ram0 initrd=0x802000000,9M'
> saveenv
```

If the size of the file system image is greater than 9M, update 9M to the correct size.

Next, type the following command to install boot monitor code.

```
> mon_install 0x0c5f0000
```

Now boot Linux using the u-boot command below.

```
> bootm 0x88000000 - 0x87000000
```

A sample SMP Linux boot log is shown below for reference

```
## Started boot kernel successfully
TCI6638 EVM # bootm 0x88000000 - 0x87000000
## Booting kernel from Legacy Image at 88000000 ...
   Image Name:   Linux-3.6.6-rt17-01071-g898535d-
   Created:      2013-01-28  16:17:47 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3215872 Bytes = 3.1 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
   Booting using the fdt blob at 0x87000000
   Loading Kernel Image ... OK
OK
   Loading Device Tree to 87ff8000, end 87fff2ea ... OK
```

```
Starting kernel ...
```

```
>>>> skern_poweron_cpu >>>>
Message2 from Secure Mode
>>>> skern_poweron_cpu >>>>
Message2 from Secure Mode
>>>> skern_poweron_cpu >>>>
Message2 from Secure Mode
[    0.000000] Booting Linux on physical CPU 0
[    0.000000] Linux version 3.6.6-rt17-01071-g898535d-dirty (a0868495@ares-ubun
tu.am.dhcp.ti.com) (gcc version 4.7.2 20120701 (prerelease) (crosstool-NG linaro
-1.13.1-2012.07-20120720 - Linaro GCC 2012.07) ) #1 SMP PREEMPT Mon Jan 28 11:17
:38 EST 2013
[    0.000000] CPU: ARMv7 Processor [412fc0f4] revision 4 (ARMv7), cr=10c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] Machine: KeyStone2, model: Texas Instruments Keystone 2 SoC
[    0.000000] cma: CMA: reserved 16 MiB at 86c00000
[    0.000000] Memory policy: ECC disabled, Data cache writealloc
[    0.000000] On node 0 totalpages: 32768
[    0.000000] free_area_init_node: node 0, pgdat c05ff440, node_mem_map c062b00
0
[    0.000000]   Normal zone: 256 pages used for memmap
[    0.000000]   Normal zone: 0 pages reserved
[    0.000000]   Normal zone: 32512 pages, LIFO batch:7
[    0.000000] PERCPU: Embedded 8 pages/cpu @c0735000 s11776 r8192 d12800 u32768
[    0.000000] pcpu-alloc: s11776 r8192 d12800 u32768 alloc=8*4096
[    0.000000] pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pag
es: 32512
[    0.000000] Kernel command line: console=ttyS0,115200n8 debug earlyprintk  rd
init=/bin/ash rw root=/dev/ram0 initrd=0x85000000,9M
[    0.000000] PID hash table entries: 512 (order: -1, 2048 bytes)
[    0.000000] Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
[    0.000000] Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
[    0.000000] Memory: 128MB = 128MB total
```

```
[    0.000000] Memory: 97800k/97800k available, 33272k reserved, 0K highmem
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
[    0.000000]     vmalloc : 0xc8800000 - 0xff000000   ( 872 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xc8000000   ( 128 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc0596c08   (5692 kB)
[    0.000000]       .init : 0xc0597000 - 0xc05cbe00   ( 212 kB)
[    0.000000]       .data : 0xc05cc000 - 0xc0601ea8   ( 216 kB)
[    0.000000]        .bss : 0xc0601ecc - 0xc062a1cc   ( 161 kB)
[    0.000000] SLUB: Genslabs=11, HWalign=64, Order=0-3, MinObjects=0, CPUs=4, N
odes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] main_pll_clk rate is 983040000, postdiv = 2, pllm = 15,plld = 0
[    0.000000] tci6614-timer: no matching node
[    0.000000] arch_timer: found timer irqs 29 30
[    0.000000] Architected local timer running at 166.66MHz.
[    0.000000] Switching to timer-based delay loop
[    0.000000] sched_clock: 32 bits at 166MHz, resolution 6ns, wraps every 25769
ms
[    0.000000] Console: colour dummy device 80x30
[    0.000092] Calibrating delay loop (skipped), value calculated using timer fr
equency.. 333.33 BogoMIPS (lpj=1666666)
[    0.000113] pid_max: default: 4096 minimum: 301
[    0.000364] Mount-cache hash table entries: 512
[    0.008205] CPU: Testing write buffer coherency: ok
[    0.008399] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.008435] hw perfevents: enabled with ARMv7 Cortex-A15 PMU driver, 7 counte
rs available
[    0.008503] Setting up static identity map for 0x804074b0 - 0x804074e4
[    0.104891] CPU1: Booted secondary processor
[    0.104932] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.154985] CPU2: Booted secondary processor
[    0.155031] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.205092] CPU3: Booted secondary processor
[    0.205141] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.205335] Brought up 4 CPUs
[    0.205375] SMP: Total of 4 processors activated (1333.33 BogoMIPS).
[    0.232209] NET: Registered protocol family 16
[    0.247428] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.256104] hw-breakpoint: found 5 (+1 reserved) breakpoint and 4 watchpoint
registers.
[    0.256114] hw-breakpoint: halting debug mode enabled. Assuming maximum watch
point size of 4 bytes.
[    0.275902] bio: create slab <bio-0> at 0
[    0.277188] SCSI subsystem initialized
[    0.278040] usbcore: registered new interface driver usbfs
[    0.278235] usbcore: registered new interface driver hub
[    0.278478] usbcore: registered new device driver usb
[    0.282607] Switching to clocksource arch_sys_counter
[    0.313194] NET: Registered protocol family 2
[    0.313891] TCP established hash table entries: 4096 (order: 3, 32768 bytes)
[    0.314031] TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
[    0.314165] TCP: Hash tables configured (established 4096 bind 4096)
[    0.314195] TCP: reno registered
[    0.314212] UDP hash table entries: 128 (order: 0, 4096 bytes)
[    0.314237] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[    0.314570] NET: Registered protocol family 1
[    0.314914] RPC: Registered named UNIX socket transport module.
[    0.314927] RPC: Registered udp transport module.
[    0.314938] RPC: Registered tcp transport module.
[    0.314949] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.315232] Unpacking initramfs...
[    0.357371] Initramfs unpacking failed: junk in compressed archive
[    0.366886] Freeing initrd memory: 9216K
[    0.512036] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[    0.512661] NTFS driver 2.1.30 [Flags: R/O].
[    0.513335] jffs2: version 2.2. (NAND) Â© 2001-2006 Red Hat, Inc.
[    0.516214] NET: Registered protocol family 38
[    0.516738] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 2
54)
[    0.516754] io scheduler noop registered
[    0.516766] io scheduler deadline registered
[    0.517099] io scheduler cfq registered (default)
[    0.629016] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    0.631409] 2530c00.serial: ttyS0 at MMIO 0x2530c00 (irq = 309) is a 16550A
[    1.272067] console [ttyS0] enabled
[    1.282480] loop: module loaded
[    1.285920] at24 0-0050: 131072 byte 24c1024 EEPROM, writable, 1 bytes/write
[    1.294796] Generic platform RAM MTD, (c) 2004 Simtec Electronics
[    1.317849] ONFI param page 0 valid
[    1.321286] ONFI flash detected
[    1.324406] NAND device: Manufacturer ID: 0x2c, Chip ID: 0xa1 (Micron MT29F1G
08ABBDAHC), page size: 2048, OOB size: 64
[    1.335350] Bad block table found at page 65472, version 0x01
[    1.341652] Bad block table found at page 65408, version 0x01
[    1.347821] 3 ofpart partitions found on MTD device 30000000.nand
[    1.353839] Creating 3 MTD partitions on "30000000.nand":
[    1.359157] 0x000000000000-0x000000100000 : "u-boot"
[    1.365625] 0x000000100000-0x000000180000 : "params"
[    1.371894] 0x000000180000-0x000008000000 : "ubifs"
[    1.378210] davinci_nand 30000000.nand: controller rev. 2.5
[    1.385151] spi_davinci 21000400.spi: master is unqueued, this is deprecated
[    1.396627] m25p80 spi32766.0: unrecognized JEDEC id 20bb18
[    1.402144] spi_davinci 21000400.spi: Controller at 0xc8886400
[    1.411193] Initializing USB Mass Storage driver...
[    1.416251] usbcore: registered new interface driver usb-storage
[    1.422163] USB Mass Storage support registered.
[    1.427169] mousedev: PS/2 mouse device common for all mice
[    1.433274] i2c /dev entries driver
[    1.438539] usbcore: registered new interface driver usbhid
[    1.444048] usbhid: USB HID core driver
[    1.448904] oprofile: using timer interrupt.
[    1.453217] Netfilter messages via NETLINK v0.30.
[    1.457862] nf_conntrack version 0.5.0 (1928 buckets, 7712 max)
[    1.464255] ctnetlink v0.93: registering with nfnetlink.
[    1.470064] IPv4 over IPv4 tunneling driver
[    1.475125] gre: GRE over IPv4 demultiplexor driver
[    1.479929] ip_gre: GRE over IPv4 tunneling driver
[    1.485851] ip_tables: (C) 2000-2006 Netfilter Core Team
[    1.491305] ipt_CLUSTERIP: ClusterIP Version 0.8 loaded successfully
[    1.497642] arp_tables: (C) 2002 David S. Miller
[    1.502261] TCP: cubic registered
[    1.505545] Initializing XFRM netlink socket
[    1.510969] NET: Registered protocol family 10
[    1.516455] NET: Registered protocol family 17
[    1.520858] NET: Registered protocol family 15
```

```
[    1.525257] 8021q: 802.1Q VLAN Support v1.8
[    1.529761] sctp: Hash tables configured (established 4096 bind 4096)
[    1.536950] NET: Registered protocol family 40
[    1.541512] rpmsg_proto_init: registered rpmsg network protocol
[    1.547471] VFP support v0.3: not present
[    1.551440] Registering SWP/SWPB emulation handler
[    1.564498] Freeing init memory: 208K
/bin/ash: can't access tty; job control turned off
/ #
```

Design/Implementation Notes

U-Boot

The MCSDK U-boot is based on the Universal Boot Loader (U-boot) public project. The documentation is available on http://www.denx.de/wiki/U-Boot/
The release is based on upstream v2013.01 U-boot release. This section gives information specific to the TCI6638EVM board and drivers only.

Board Support

The Keystone EVM board file islocated at board/ti/ks2_evm (for MCSDK 3.0.x, it is located at board/ti/tci6638_evm directory). It has PLL, DDR3 configurations and initialization functions.

SoC Support

The Keystone chip specific code is located at arch/arm/cpu/armv7/keystone directory. It has the following files:

- aemif.c – Asynchronous EMIF configuration.
- init.c – chip configuration
- clock.c – clock related functions
- cmd_clock.c, cmd_mon.c, cmd_fmtimg.c – implementation of “psc”, “getclk”, “pllset”, “fmtimg” and “mon_install” shell commands.
- psc.c – PSC driver.
- cppi_dma.c – simple CPPI_DMA driver.

Drivers

The Keystone EVM uses the following drivers:

- SPI - drivers/spi/davinci_spi.c.
- I2C - drivers/i2c/keystone_i2c.c.
- UART - serial.c, ns16550.c and serial_ns16550.c at drivers/serial directory.
- NAND - drivers/mtd/nand/davinci_nand.c
- ETH - drivers/net/keystone_net.c
- USB - generic xhci support from patches patchwork.ozlabs.org/patch/193476/ and patchwork.ozlabs.org/patch/193477/. Platform specific xhci support in drivers/usb/host/xhci-keystone.c

Ethernet Driver

The K2HK EVM has 4 Ethernet ports (port 0 - 3) with K2HK_EMAC, K2HK_EMAC1, K2HK_EMAC2, K2HK_EMAC3 interface names. The K2L EVM has 4 Ethernet ports (port 0 - 3) with K2L_EMAC0, K2L_EMAC1, K2L_EMAC2, K2L_EMAC3 interface names. The "ethact" environment variable selects the currently active Ethernet interface. For example:

```
set ethact K2HK_EMAC1
```

selects the port 1 to be used for following network commands.

You may change the active interface on runtime.

The **ethact** variable is set on start of u-boot to the name of the first registered interface. The TCI6638_EMAC is registered first and therefore ethact is always set to that name. You may want to use another environment variable to set desirable interface. You can do that extending init_${boot} variables.

MAC address of the port 0 is taken from SOC e-fuse. You may want to overwrite it setting the “ethaddr” environment variable. The SoC doesn’t have an e-fuse for second port MAC address. You have to set the “eth1addr” explicitly for that port.

NOTE: U-boot does not allow to change the "ethaddr" variable if it is already set. If you want to change it you need to use the "env default -a" command, which resets all environment variabels to default values and deletes the "ethaddr" variable as well.

The board_k2x.c file has the eth_priv_cfg array, which has the default configuration of the both interfaces.

```
eth_priv_t eth_priv_cfg[] = {
        {
                .int_name       = "K2HK_EMAC",
                .rx_flow        = CPSW_PORT_RX_FLOW(0),
                .phy_addr       = 0,
                .slave_port     = 1,
                .sgmii_link_type = SGMII_LINK_MAC_PHY,
        },
        {
                .int_name       = "K2HK_EMAC1",
                .rx_flow        = CPSW_PORT_RX_FLOW(1),
                .phy_addr       = 1,
                .slave_port     = 2,
                .sgmii_link_type = SGMII_LINK_MAC_PHY,
        },
        {
                .int_name       = "K2HK_EMAC2",
                .rx_flow        = CPSW_PORT_RX_FLOW(2),
                .phy_addr       = 2,
                .slave_port     = 3,
                .sgmii_link_type = SGMII_LINK_MAC_MAC_FORCED,
        },
        {
                .int_name       = "K2HK_EMAC3",
                .rx_flow        = CPSW_PORT_RX_FLOW(3),
```

```
                .phy_addr       = 3,
                .slave_port     = 4,
                .sgmii_link_type = SGMII_LINK_MAC_MAC_FORCED,
        },
};
```

where

- "phy_addr" is the phy address on mdio bus.
- "slave_port" is the port number starting from "1"
- "sgmii_link_type" - type of SGMII link.

The emac_def.h defines possible sgmii link types

```
#define SGMII_LINK_MAC_MAC_AUTONEG      0
#define SGMII_LINK_MAC_PHY              1
#define SGMII_LINK_MAC_MAC_FORCED       2
#define SGMII_LINK_MAC_FIBER            3
#define SGMII_LINK_MAC_PHY_FORCED       4
```

By default Ethernet driver assumes that phys are not connected to MDIO bus. Using the “has_mdio” environment variable you may override this default:

```
“setenv has_mdio 1” – tells the driver that PHYs are connected to MDIO.
```

If has_mdio=0 or there is no such environment variable the sgmii_link_type for each port is set to SGMII_LINK_MAC_PHY_FORCED. Be aware that the sgmii_link_type for each interface may be overridden by setting the sgmiiN_link_type environment variable. For example:

```
“setenv sgmii1_link_type 2” -  sets the SGMII_LINK_MAC_MAC_FORCED link type for port 1.
```

Driver doesn't perform sanity check of the settings. It is your responsibility to set correct values.

The K2 SOC has 4 SGMII port and you can add configurations for port2 and port3 to the eth_priv_cfg[]. Though driver supports four ports, as K2K EVM has only 2 ports the port2 and port3 functionality wasn’t tested.

Network driver supports Multicast tftp. To enable this feature, add following in the respective config.h file (example include/configs/k2hk_evm.h for K2HK)

```
#define CONFIG_MCAST_TFTP
```

10G Ethernet Driver

K2HK and K2E supports 10G ethernet connections through a 3-port (1 host and 2 slaves) 10G ethernet switch. On K2HK and K2E EVMs, the physical 10G ethernet connection is brought out to a Rear Transition Module-Break Out Card (RTM-BOC). This implementation **supports only the Rev.C RTM-BOC with Dual Retimers**.

The U-boot implementation of the 10G ethernet driver can be considered as an extension of the 1G ethernet driver framework. It adds two interfaces to the data structure eth_priv_cfg[] in the corresponding board_k2x.c files:

```
eth_priv_t eth_priv_cfg[] = {
        ...
        {
                .int_name       = "K2HK_XMAC0",
                .rx_flow        = 0,
                .slave_port     = 1,
                .sgmii_link_type = XGMII_LINK_MAC_MAC_FORCED,
        },
        {
                .int_name       = "K2HK_XMAC1",
                .rx_flow        = 8,
                .slave_port     = 2,
                .sgmii_link_type = XGMII_LINK_MAC_MAC_FORCED,
        },
};
```

with a new interface sgmii_link_type of XGMII_LINK_MAC_MAC_FORCED. The interface names are K2HK_XMAC0 and K2HK_XMAC1 on K2HK; and K2E_XMAC0 and K2E_XMAC1 on K2E.

From the user perspective, there is no difference in configuring and using the 10G interfaces except that there is no has_mdio or sgmii_link_type setenv settings. See Ethernet Driver for more configuration details.

When U-boot boots up, it will only show the 10G interface names if the proper RTM-BOC is connected to the EVM. If so, they are the 4th and 5th (0-based) interfaces on K2HK; and 8th and 9th on K2E. Hence to properly configure the corresponding MAC addresses, eth4addr and eth5addr on K2HK should be used, while they are eth8addr and eth9addr on K2E .

I2C Driver

The keystone_i2c.c driver supports multiple I2C buses. Thus K2 SoC has three buses 0, 1 and 2. Only one bus can be uses at the same time. Use the

```
int i2c_set_bus_num(unsigned int bus)
```

function to set desired bus number. You may get the current bus number by calling the

```
unsigned int i2c_get_bus_num(void)
```

function.

In order to set/get current bus number from u-boot shell use “i2c dev” command.

```
TCI6638 EVM # i2c dev
Current bus is 0
TCI6638 EVM # i2c dev 1
Setting bus to 1
TCI6638 EVM #
```

NAND Driver

The NAND driver used is davinci_nand.c. For K2L, NAND part is 2G vs other EVMs that has 512M part. The EVM by default create 3 partitions.

1. bootloader 2. params (env) 3. ubifs. This has boot volume to hold the images and rootfs to volume to hold the file system

UBI boot time is dependent on the ubifs partition size. So on K2L since the size is much bigger, it takes more time because the mount is done on a bigger nand parition. To reduce the boot time customer may resize the partition to a smaller to hold the file system. Additional partitions can be defined to use as data storage or other applications. Same change needs to be reflected in the DTS as well as mtdparts env variable in u-boot.

SPL

SPL stands for Secondary Program Loader. This is a framework added to unpstream U-Boot in version 2012-10. SPL is supported in U-Boot for keystone II devices. This feature is configured using the config option CONFIG_SPL. It allows creation of a small first stage boot loader (SPL) that can be loaded by ROM Boot loader (RBL) which would then load and run the second stage boot loader (full version of U-Boot) from NOR or NAND. The required config options are added to tci6638_evm.h. User may refer the README text file in the U-Boot root source directory for more details.

For keystone II, RBL loads SPL image from offset 0 of the SPI NOR flash in spi boot mode. The first 64K of the SPI NOR flash is flashed with SPL followed by u-boot.img. The initial 64K is padded with zeros.

Thump mode

U-Boot build uses ARM thump mode. This is configured using CONFIG_SYS_THUMB_BUILD option.

Power-On Self-Test (POST)

POST will perform the following functional tests:

- EEPROM read test
- NAND read test
- External memory read/write test

By default the POST is enabled. Set "no_post" environment variable to disable it.

```
set no_post 1
```

Note: If needed to disable POST on next reboot, this variable can be saved using saveenv.

FDT command

Using FDT commands, user will be able to tweak the FDT before boot to Linux. An example of this command usage to enable the PSC module and Power domain for SRIO and Hyperlink IP is discussed here. The device tree bindings for the clock node for these IPs (functional blocks) have the status property defined and set to "disabled" by default. If user wants to enabled this so that user space drivers can be used for these IPs, the status property can be updated in the DTB blob through fdt command before boot to Linux. Following env variables can be set in u-boot to achieve this:-

```
setenv modify_fdt 'fdt addr ${addr_fdt}; fdt set /soc/clocks/clkhyperlink0
status "enabled"; fdt set /soc/clocks/clkhyperlink1 status "enabled"; fdt
set /soc/clocks/clksrio status "enabled";'
setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot}
modify_fdt run_mon run_kern'
```

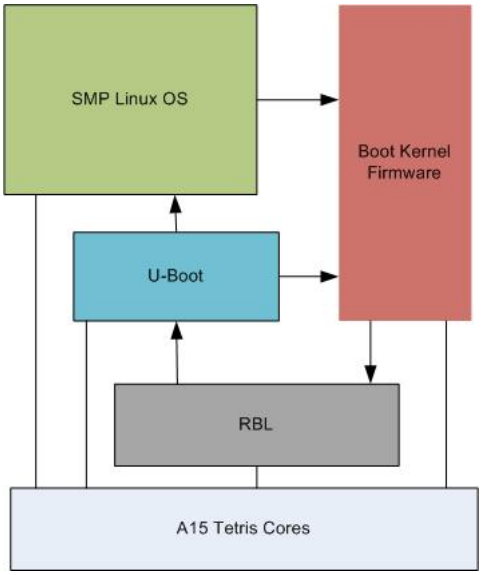
User may modify the above based on which IP is to be enabled. In the above example, two hyperlink and one srio IPs are enabled (LPSC module is enabled as well the Power Domain). This command will be useful to fix up DT bindings in the DTB before booting to Linux.

Reserve DDR memory

U-Boot has mem_reserve env variable to reserve DDR3 memory at the end of the 32 bit address space. This will be useful for reserving memory for DSP. Based on the memory availability on the board, the address range of this region will change. So any users of this feature need to make sure the address match with what is reserved through this mechanism. Otherwise the user application can step into kernel memory space and cause kernel crash during system operation. By default 512M memory is reserved at the end of the address space. To change the default size, user need to update this env variable and save the configuration using saveenv command.

Boot Monitor

Boot Monitor software provides secure privilege level execution service for Linux kernel code through SMC calls. ARM cortex A15 requires certain functions to be executed in the PL1 privilege level. Boot monitor code provides this service. A high level architecture of the boot moinitor software is shown below



Boot monitor code is built as a standalone image and is loaded into MSMC SRAM at 0xc5f0000 and occupies top 64K of the memory. The image has to be loaded to the at the above address either through CCS or through tftp or other means. It gets initialized through the u-boot command install_skern. The command takes the load address above as the argument.

Boot sequence of primary core

In the primary ARM core, ROM boot loader (RBL) code is run on Power on reset. After completing its task, RBL load and run u-boot code in the non secure mode. Boot monitor gets install through the command mon_install(). As part of this following will happen

- boot monitor primary core entry point is entered via the branch address 0xc5f0000
- As part of non secure entry, boot monitor calls the RBL API (smc #0) through SMC call passing the _skern_init() as the argument. This function get called as part of the RBL code
- _skern_init() assembly function copies the RBL stack to its own stack. It initialize the monitor vector and SP to point to its own values. It then calls skern_init() C function to initialize to do Core or CPU specific initialization. r0 points to where it enters from primary core or secondary core, r1 points to the Tetris PSC base address and r2 points to the ARM Arch timer clock rate. RBL enters this code in monitor mode. skern_init() does the following:-
- Initialize the arch timer CNTFREQ
- Set the secondary core non secure entry point address in the ARM magic address for each core
- Configure GIC controller to route IPC interrupts

Finally the control returns to RBL and back to non secure primary core boot monitor entry code.

- On the primary core, booting of Linux kernel happens as usual through the bootm command.
- At Linux start up, primary core make smc call to power on each of the secondary core. smc call is issued with r0 pointing to the command (0 - power ON). r1 points to the CPU number and r2 to secondary core kernel entry point address. Primary core wait for secondary cores to boot up and then proceeds to rest of booting sequence.

Boot sequence of secondary cores

At the secondary core, following squence happens

- On power ON reset, RBL initializes. It then enters the secondary entry point address of the boot monitor core. The init code calls smc #0 to invoke _skern_init() as a privilege function and in turn sets its own stack, vectors and smc service functions. _skern_init() calls skern_init() C function to initialize per CPU variables. It initialize the arch timer CNTFREQ to desired value.
- On return from _skern_init(), it jumps to the secondary kernel entry point address and start booting secondary instance of Linux kernel.

LPAE

The DDR3A memory address space is 0x800000000-0x9FFFFFFFF. That range is outside the first 4GB of address space. Even if Cortex-A15 supports Large Physical Address Extension (LPAE), which allows 40 bits address bus, and may access the DDR3A address space, that is possible only when ARM turns MMU on. Before that ARM has 32-bits effective address bus and can access only first 4GB of address space.

In order to give access to the DDR3A when MMU is off, MSMC maps first 2GB of DDR3A to the 0x00800000000-0x00FFFFFFFF address space. That aliased address space of the first 2GB of the DDR3A may be used not only when MMU is off, but when it is on as well. Be aware that Keystone2 doesn't support cache coherency for the aliased address space.

The memory property in the tci6638-evm.dts file specifies the memory range available for kernel. The default sets 0x80000000 as start address and 0x20000000 as size. That corresponds to the aliased DDR3A memory range.

When u-boot boots the kernel, it can modify the start address of the "memory" in the DTB. If the "**mem_lpae**" environment variable is "0", u-boot doesn't modify the start address. If the **mem_lpae=1**, u-boot sets the start address to 0x800000000, which corresponds to non-aliased start address of the DDR3A.

Using more than 2GB of DDR3A memory

U-boot works without MMU enabled and its address range is limited to the first 4GB. The 2GB of that range is the DDR3A aliased address range. Therefore u-boot without MMU enabled cannot detect more than 2GB of DDR3A size. Kernel works with MMU enabled and when LPAE is enable may use up to 8GB of the DDR3A memory.

To pass the memory size information to the kernel u-boot fixes the memory node of the DTB. If the SO-DIMM size is less or equal 2GB u-boot creates memory node with one bank only:

```
memory {
        reg = <0x00000008 0x00000000 0x00000000 0x80000000>;
};
```

U-boot may reserve part of the memory at the beginning or end of the first 2GB. If the "mem_reserve_head" variable is set, u-boot modifies the start address and the size of the memory bank on the mem_reserve_head. For example if the mem_reserve_mem=256M u-boot creates the following memory node:

```
memory {
        reg = <0x00000008 0x10000000 0x00000000 0x70000000>;
};
```

If the mem_reserve variable is set, u-boot reserves the memory at the end of the first bank. For example if the mem_reserve=512M u-boot creates the following memory node:

```
memory {
        reg = <0x00000008 0x00000000 0x00000000 0x60000000>;
};
```

If the board has SO-DIMM bigger than 2GB user has to set the ddr3a_size environment variable. The variable may be set to 4 or 8. If u-boot detects the 2GB of memory (maximum it may detect), it checks the "mem_lpae" and "ddr3a_size" variables. If the mem_lpae=1 and ddr3a_size=4 or 8, u-boot creates the memory node with two banks. The first bank represents first 2GB of memory with possible reserved memory. The second bank represent the remaining memory. That memory starts at the fixed address and size equal "ddr3a_size - 2".

For example if mem_lpae=1, mem_reserve=512M and ddr3a_size=8 u-boot creates the following memory node:

```
memory {
        reg = <0x00000008 0x00000000 0x00000000 0x60000000
               0x00000008 0x80000000 0x00000001 0x80000000>;
};
```

It is important that u-boot creates two memory banks. That allows to reserve memory at the first bank (first 2GB of memory). There are SOC masters which have only 32 bit address bus and may access the first 2GB of the DDR3A only.

It is possible to reserve memory at the second bank. A customer has to modify the u-boot ft_board_setup() function to do that.

DDR3 ECC

DDR3 error detection and correction feature is enabled on K2H/K2K PG 2.0 devices and K2L/K2E devices. The DDR3 controller supports ECC on the data written or read from the SDRAM and is enabled by programming the ECC Control register. 8-bit ECC is calculated over 64-bit data quanta. The ECC is calculated for all accesses that are within the address ranges protected by ECC. 1-bit error is correctable by ECC and 2-bit error is not correctable and will be treated as unrecoverable error by sotware and trigger the reset of the device.

DDR3 ECC Handling in U-boot

U-boot checks if the DDR3 controller supports ECC RMW or not. If ECC RMW is not supported (in K2H/K2K PG1.x devices), U-boot will disable the ECC by default, otherwise it always enables ECC (in K2H/K2K PG2.0 devices and K2L/K2E devices)

During the ECC initialization, U-boot fills the entire memory (up to 8GB) to zeros using an EDMA channel after ECC is enabled. For K2H/K2K/K2L device, U-boot configures the chip level interrupt controller to route the DDR3 ECC error interrupt to ARM interrupt controller. For K2E device, since DDR3 ECC error interrupt is directly routed to ARM interrupt controller, there is no need to configure the chip level interrupt controller.

A new DDR3 command is added to simulate the ECC error, the command format is:

```
ddr ecc_err <addr in hex> <bit_err in hex> - generate bit errors in DDR data at <addr>, the command will read a 32-bit data from <addr>, and write (data ^ bit_err) back to <addr>

E.g.:
ddr ecc_err 0x90000000 0x1 (this will genereate a 1-bit error on bit 0 of the data in ddr address 0x9000_0000)
ddr ecc_err 0xa0000000 0x1001 (this will genereate 2-bit error on bit 0 & 3 of the data in ddr address 0xa000_0000)
```

A new environment variable "ecc_test" is also introduced to test ECC. By default, ecc_test = 0, and any detection of 2-bit error will reset the device. If ecc_test = 1, U-boot will bypass the error and continues to boot Linux kernel so that Linux kernel can handle the error in interrupt service.

DDR3 ECC Handling in Linux kernel

Linux kernel requests an IRQ handler for DDR3 ECC error interrupt, the handler checks the DDR3 controller interrupt status register, if the error is 2-bit error, Linux kernel will reboot the device. User can also use a user mode command to read the DDR3 ECC registers (e.g. 1-bit error count register, etc.), the DDR3 controller register and interrupt mapping are defined in the sysctrl node of device tree binding:

```
E.g. K2HK SOC device tree:
sysctrl {
        reg = <0x21010000 0x0200>; /* DDR3 controller reg */
        interrupts = <0 24 0xf01    /* L1L2 ECC error interrupt */
                      0 448 0xf01>; /* DDR3 ECC error interrupt */
};
```

Linux Kernel

The Keystone II devices runs SMP Linux and is based on upstream Linux version. The Linux port uses arch/arm/mach-keystone for the machine specific intialization code. Linux is boot up through U-boot on the primary ARM core. Linux depends on the Boot Monitor software that gets installed through U-Boot for SMP boot up. Please refer the Boot Monitor section for more details on Linux boot up sequence.

Device Tree bindings are used to define the configuration of the board to use for Linux boot up. This makes it easy for users to define bindings for their custom board so that same image provided in the release may be used to run on their board. Note that any additional peripheral drivers required on the custom board is ther responsobility of the user. User will be able to use the Device Tree bindings to configure the driver. The traditional machine setup code based device initialization support is not available for Keystone devices.

SoC Support

SoC specifc initialization and setup code is located under arch/arm/mach-keystone folder. Here are the basic init done for Keystone II devices.

- pmc.c - Power Management Run time init code
- platsmp.c - Provides smp_operations for SMP Linux operation. This is where code to Power ON secondary cores resides.

- keystone.c - DT_MACHINE_START macro for Keystone I and Keystone II devices are defined in this file. This also does clock, arch timers and Device Tree specific initialization. It provides machine specific restart() function for Linux.

GIC irq chip driver

gic irqchip driver is re-used for the Keystone II devices. This the parent irq controller. Most of the peripheral driver bindings include phandle for this irq controller. This driver is currently located under arch/arm/common folder and is moved to drivers/irqchip folder in recent upstream kernel releases. Device bindings for the driver can be found in Linux kernel tree under Documentation/arm/gic.txt

Keystone IPC irq chip driver

This is the irq controller driver for receiving IPC interrupts from DSPs. It manages the IPCGRx and IPCARx registers for the Host. It uses GIC irq controller as the parent. The bindings are similar to that for GIC. However it uses "interrupts" property in the bindings to define the parent interrupt line to the GIC interrupt controller. Two cells are used to identify an interrupt line to IPC IRQ chip. Up to 28 interrupt lines (Logical) are terminated on this device and maps to bits of the above IPC registers. One user of this driver is the remote proc user driver that uses these interrupts to receive interrupts from DSPs. The bindings for rproc user driver provides "interrupts" property to identify the specific interrupts terminated on this device.

SMP

Boot Setup

SMP Linux boot support is implemented in U-Boot, Boot Monitor and Linux. In the U-Boot, a command mon_install() is added to allow initialization of boot monitor. This command is to be included in bootcmd as part of U-Boot environment setup. This command assumes that boot monitor image is loaded in MSMC Memory prior to the invokation of the command. More details on Boot Monitor is provided elsewhere in this document. Linux platform code uses the SMC calls to inoke monitor services provided by boot monitor, for example smc call to Power ON secondary core is one such service.

Common Clock framework

In this release, all of the clock hardware nodes in the platform are represented in device tree. Here is the high level view of the clock tree

```
refclkmain (fixed clock)
  - mainpllclk (Main PLL clock)
    - mainmuxclk (Mux clock)
       - chipclk1 (chip fixed factor clocks)
          - clkusb (PSC clocks)
```

Please refer the Device data manual for the description of various clocks available on the SoC. The devices gets the Main PLL input clock from the external source. The refclkmain represents this fixed clock. Main PLL output clock is represented bu mainpllclk node. The output of this clock is fed to the mainmuxclk which can either select refclk directly (bypassing the Main PLL) or the output of the Main PLL to pass to the next stage which are divider clocks. The SYSCLKx are the divider clocks that then gets distributed to various IPs. The PSC clocks represents the LPSC (local power sleep controller) availale for individual IPs.

The driver for each of the above clock device is shown below

1. fixed clock - example refclkmain

```
drivers/clk/clk-fixed-rate.c
```

2. Main PLL clock - example mainpllclk

```
drivers/clk/clk-keystone-pll.c
```

3. (Mux clock) - example mainmuxclk

```
drivers/clk/clk-mux.c
```

4. fixed factor clocks (example chipclk1)

```
drivers/clk/clk-fixed-factor.c
```

5. Psc clocks (example - clkusb)

```
drivers/clk/davinci/clk-davinci-psc.c
```

6. Common clock init code

```
drivers/clk/davinci/davinci-clock.c
```

More details on common clock framework is part of Linux kernel documentation at Documentation/clk.txt Device clock bindings are documented at Documentation/devicetree/bindings/clock/*

The common clock code is enable through the CONFIG_COMMON_CLK KConfig option.

There are debugfs entries for each clock node to display the rates, usecounts etc. This replace the old debugfs entry to display the clock information. For example to show the main PLL clock rate, use the following command

```
root@tci6614-evm:~# mount -t debugfs debugfs /debug
root@tci6638-evm:~# cat /sys/kernel/debug/clk/refclk-main/mainpllclk/clk_rate
798720000
```

AEMIF Driver

DaVinci AEMIF driver is re-used for Keystone devices. The driver is implemented in drivers/memory/davinci-aemif.c. The binding definitions are given in Documentation/devicetree/bindings/memory/davinci-aemif.txt. This driver allows configuration of the AEMIF bus connected to slave devices such as NAND. The bindings exposes the timing parameters required for the slave device.

NAND Driver

Davinci NAND controller driver is re-used for the Keystone devices. The required bindings are defined under Documentation/devicetree/bindings/mtd/davinci-nand.txt. Driver file is drivers/mtd/nand/davinci_nand.c

SPI and SPI NOR Flash Drivers

DaVinci SPI controller driver is re-used for keystone devices. Driver is implemented in drivers/spi/spi-davinci.c and the bindings in Documentation/devicetree/bindings/spi/spi-davinci.txt. The SPI NOR flash driver used is m25p80.c under drivers/mtd/devices/m25p80.c. The bindings for this driver is available under Documentation/devicetree/bindings/mtd/st-m25p.txt

I2C and EEPROM Drivers

DaVinci I2C controller driver is re-used for Keystone devices. The driver file is drivers/i2c/chip/i2c-davinci.c and the binding is defined in Documentation/devicetree/bindings/i2c/davinci.txt. EEPROM driver is a i2c client driver and is implemented in drivers/misc/eeprom/at24.c. The binding for this is also part of the davinci i2c driver bindings.

Keystone GPIO Driver

GPIO driver is implemented in drivers/gpio/gpio-keystone.c. Each instance of the driver support 32 GPIO pins (2 Banks). Currently only 2 banks are tested on keystone devices. The driver also implements gpio irq chip driver that exposes gpio pin as irq pins and maps them to the corresponding irq pins input of GIC. This driver allow user drivers to define the GPIO pins that it uses as bindings and points to the bindings of this device. Also allows user driver to use a GPIO pin as interrupt. The irq chip driver supports upto 32 irq pins. Driver uses the pin as irq or GPIO, not both. Please refer to bindings documenation at Documentation/devicetree/gpio/gpio-keystone.txt for details of usage. Note that user driver can also use the GPIO pins using the GPIO LIB APIs. User needs to choose either defining the GPIOs through bindings or using API, and not both for a specific pin.

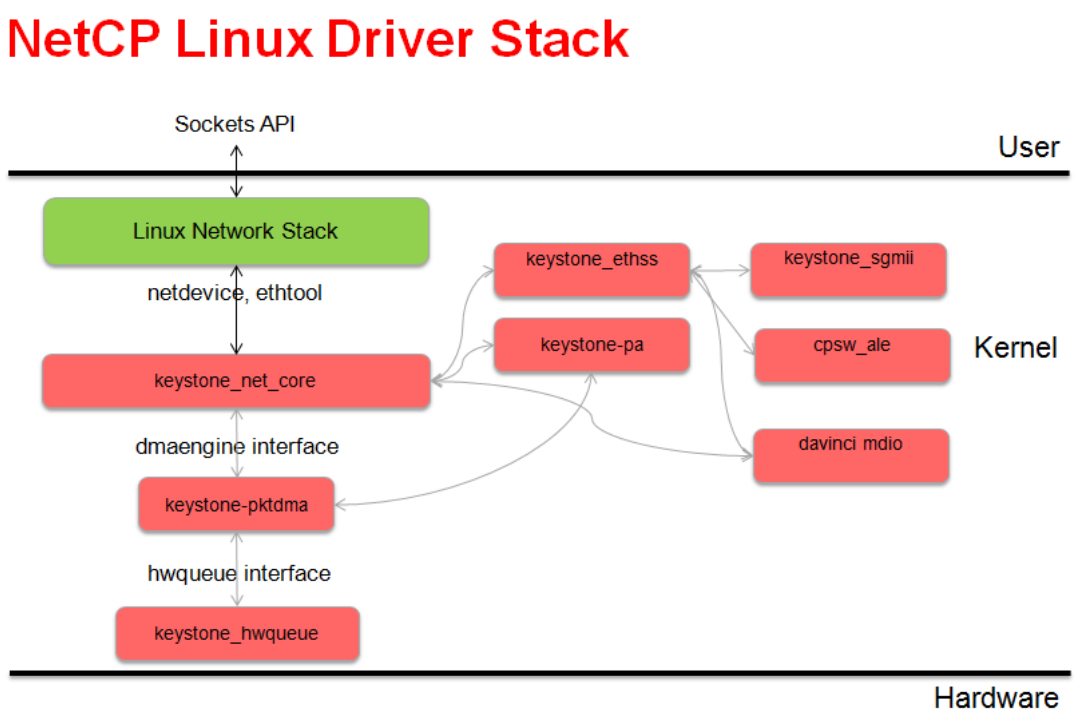
Keystone IPC GPIO Driver

Keystone IPC GPIO driver exposes IRQs to DSPs (using IPCGRx register of each DSP) as GPIO pins. Software users allocates a GPIO pin using GPIO lib API or using a gpio bindings in the driver using phandle to the corresponding gpio ipc device binding. This driver allows defining one instance of the GPIO driver per DSP and allows up to 28 GPIO pins per GPIO instance. The bindings for each instance is defined in the dts file for the platform. The driver is implemented in drivers/gpio/gpio-keystone-ipc.c. Documentation/devicetree/bindings/gpio/gpio.txt has description on how to define bindings in the user driver to use a GPIO pin. The user driver uses GPIO lib API calls to write a value of 1 to the GPIO output pin that will raise IRQ to the DSP.

Watchdog Driver

The Watchdog driver is implemented in drivers/watchdog/davinci_wdt.c. For an explanation on the device tree bindings required see Documentation/devicetree/bindings/watchdog/davinci-wdt.txt.

Network Driver



- The NETCP driver code is at **/drivers/net/ethernet/ti/keystone_net_core.c** in the linux kernel.

Model:

1) Open:

On device open, we request DMA channels using standard dma engine APIs provided by the dmaengine layer.
The capabilities of the DMA are appropriately set and the appropriate channels are requested by name.
We invoke the dma_request_channel_by_name API to acquire dma engine channels.
The netdev open also sets up the streaming switch, SGMII, Serdes, switch and mac sliver. On Open we also configure the packet accelerator module.

2) Stop:

A standard netdev stop operation that stops the netif queue.
It sets the receive state to teardown and calls dmaengine_pause APIs from the dmaengine layer to pause both the RX and TX channel.
After NAPI is disabled, the operation goes onto release the dma channels by calling the dma_release_channel API.
The RX state is set to invalid and the ethernet susbsystem is stopped

3) Polling

A NAPI poll function is implemented, which sets the RX state to poll and resume the dma engine.
It will then go onto initialize the queues. We allocate memory for each packet and initialize the scatter list.
We iterate in a loop till we run out memory for the descriptors. These dma descriptors are acquired using a dmaengine API called device_prep_slave_sg.
We pass appropriate arguments to the API. At this instant we are not concerned about the Software info and PS info associated with each descriptor.
netcp_rx_complete is the callback associated with the dma engine on the RX side.
The callback function checks for the correct DMA and RX state and warns the user on seeing abnormal behavior.
It then goes onto stahs the received packet to the tail of a linked list.

4) Xmit

netcp_ndo_start_xmit is the standard netdev operation that is used to push an outgoing packet.
Again we have a structure for each packet. This will contain an array of scatterlists and the number of scatterlist entries which at this point should be 1.
We then call the device_prep_slave_sg API with appropriate parameters to acquire a descriptor for the TX operation.
'netcp_tx_complete' is the callback fucntion attached to each TX dma callback.
All the network statistics are appropriately updated on a successful transfer. The callback then proceeds to free the skb with the dev_kfree_skb_any API.

5) Receive

In the napi_poll netdev operation, we call the netcp_refill_rx function which will allocate skbs and attch these skbs to a descriptor until we run out of descriptor memory. the deliver_stash routine fills the appropriate RX

DMA Engine

The PktDMA interface is an extension of the Linux DMA Engine subsystem.
It is worth reviewing the following documentation found in the kernel source tree

- Documentation/dmaengine.txt
- Documentation/devicetree/bindings/dma/keystone-pktdma.txt

Device Tree Requirements

Each PktDMA channel must be defined in the device tree prior to use.
The device tree channel description associates the channel with one or more hardware queues that must also be described in the device tree.
The PktDMA code generally identifies a device tree description by the channel label string.
For example, the Keystone Ethernet driver uses two channels, named “nettx” and “netrx”, used for outbound and inbound traffic respectively.
Transmit channels must define the “submit-queue” tag, while receive channels must define the “flow” tag.

DMA Channel Configuration

First, open a DMA channel by calling dma_request_channel_by_name(), passing in a mask requesting the DMA_SLAVE
capability and the desired channel name.
The returned pointer is used as a handle for all subsequent operations on the channel.
Note that while a NULL pointer indicates an error, there are non-NULL values that also indicate errors;
use the IS_ERR_OR_NULL() macro to test the returned value. <The next step is to configure the channel using
the dma_keystone_config() function, which takes a dma_keystone_info structure.
The fields that must be filled in depend on whether the channel is to be used as a transmit or receive channel.
In both cases, all unused fields must be zero.

For a transmit channel, set these fields:

- Set the direction field to DMA_MEM_TO_DEV.
- Set the tx_queue_depth field to the the number of transmit buffers that can be outstanding at any one time.

Attempting to prepare a buffer or chain of buffers for transmission when there is insufficient depth remaining will result in an error.

For a receive channel, set these fields:

- Set the direction field to DMA_DEV_TO_MEM.
- Set the scatterlist_size field to the number of entries in the array of scatterlist structures that will be used for reception. If the application does not support scatter/gather, then this will be between 1 and 3.
- Set the rxpool_allocator field to be a pointer to a function that the DMA engine will call to replenish the pool of free receive buffers.
- Set the rxpool_destructor field to be a pointer to a function that the DMA engine will call to free the pool of free receive buffers during channel shutdown and certain error conditions.
- The rxpool_param field is a pointer to a void that is passed to the rxpool allocator and destructor functions to make them easier to write. The PktDMA subsystem makes no other use of this field.
- Set the rxpool_thresh_enable field to DMA_THRESH_NONE. In the future the PktDMA interface will support selection of free buffer queue by packet size, but this is not currently supported.
- The rxpools array determines how the four free buffer queues are configured. If neither scatter/gather nor size thresholds are being used, fill in only the first entry in the array (rxpools[0]); this is likely to be the normal case. Each entry contains a buffer size and desired pool depth. If using packet size thresholds, the array must be filled in from smallest to largest.
- Set the rxpool_count to the number of rxpools array entries being used.

The dma_keystone_config() function will handle the opening and initialization of hardware queues, and allocation of hwqueue descriptors.
For transmit channels the number of descriptors is given by the tx_queue_depth field.
For receive channels the number of descriptors is the sum of the pool_depth fields in the rxpools array.

Notification and Completion Handlers

When an outgoing packet has been transmitted, the hardware will place the descriptor on a completion queue.
Similarly, when an incoming packet has been received, the hardware places the descriptor on a receive queue.
In either case, an interrupt may be generated.
The driver can register to be notified of the interrupt by establishing a notification handler using the dma_set_notify() function.
Note that the notification handler is called from within the hardware interrupt service routine,
and thus runs in hard interrupt context with the IRQ disabled.
Only a minimum of processing should be done in this context; typically disabling further interrupts
on the channel using the dmaengine_pause() function, and scheduling deferred work in a tasklet or work queue.

When the deferred work function is invoked, it should invoke the dma_poll() function.
This will cause the PktDMA subsystem to pop packet descriptors from the completion queue and invoke the associated completion functions.
This process continues until the completion queue is empty or the “budget” of descriptors has been exhausted.
The return value from dma_poll() is the number of packets processed.

The hwqueue subsystem configures the queues with interrupts enabled.
If a notification handler is not established for a channel and interrupts remain enabled,
the packet completion functions will be called from hard interrupt context.
This is generally not desirable. It is recommended that either a notification function and a deferred work mechanism be established,
or the completion interrupt should be disabled and the dma_poll() function called periodically.

Packet Transmission Initiation

To transmit a packet, initialize an array of scatterlist structures. If the packet descriptor is to contain an Extended Packet Information Block,
this should be described by the first entry in the scatterlist array. If the descriptor is to contain a Protocol Specific data block, this should be described next.
These areas a copied into the descriptor from system virtual address space by the CPU and are not subject to DMA requirements.
The remaining entries in the scatterlist array describe the data buffer.
A data buffer for transmission must reside in an area of memory that is (1) in an address space accessible to the PktDMA hardware, and (2) will not be
modified or reallocated until the operation has completed. If the entire data area is not physically contiguous,
each physically contiguous section must be described by a separate scatterlist array entry – a so-called “gather” operation.
Once the scatterlist array is complete, the data buffer segments must be mapped for the DMA_TO_DEVICE transfer.
If it is known that there will be only one data segment this done using the dma_map_single() function, or dma_map_sg() function if there may be multiple segments.
Do not access the buffer after it has been mapped; this transfers “ownership” of the buffer to the PktDMA device and doing so will result in unpredictable behavior.
Do not map the EPIB or PSINFO segments.
After mapping the data buffer segments, pass the scatterlist array to the dmaengine_prep_slave_sg() function,
the number of scatterlist array entries used, and flags indicating whether scatterlist array includes EPIB or PSINFO segments.
If there are no problems, this function returns the address of a dma_async_tx_descriptor structure.
Each packet should be associated with a completion routine that will be called by the PktDMA subsystem after the packet has been accepted.
To do this, fill in the callback field of the dma_async_tx_descriptor structure with the address of the completion routine.
To make coding easier, the content of the callback_param field of this structure (a void pointer) is passed to the completion routine as a parameter.
The packet is now ready for transmission. Call dmaengine_submit(), passing the address of the dma_async_tx_descriptor.
The return value is a “cookie” that will be used later to check the completion status of the packet.

Packet Transmission Completion

The packet completion routine for transmission is relatively simple.
The completion status should be checked by calling the dma_async_is_tx_complete() function.
Unfortunately, this routine requires the “cookie” returned by dmaengine_submit(), which makes for a possible race condition.
The data buffers, which were mapped for DMA earlier, must be unmapped using the dma_unmap_sg() function or equivalent.

Receive Free Queue Management

Before the PktDMA hardware can transfer a received packet to the host, the receive free queue or queues must be populated.
The desired buffer sizes and depths of these queues is established when the channel is configured.
To populate the queue, call the dma_rxfree_refill() function.
This will call the rxpool_allocator function repeatedly until the pool reaches its configured depth, or until the allocator function returns a NULL pointer.
This is done for each configured queue.

The first is the value of rxpool_param set during configuration.
The second is the free queue number, an integer value between 0 and 3, indicating which free queue is being populated.
The third is the size of the buffer to be allocated, as passed in the rxpools array.
The buffer size is not used by the PktDMA code, and is provided only for the convenience of the allocator and destructor functions.

The rxpool_allocator function must return the address of a dma_async_tx_descriptor structure (or NULL).
The sequence for this is very similar to transmission, with these exceptions:

- Only single buffers may be returned, no multi-buffer chains.
- The scatterlist arrays used for preparing buffers that may be used as the first buffer in a packet (typically queue number 0) must be discrete and persistent until the buffer is returned in a completion queue or passed to the rxpool_destructor function.
- The scatterlist arrays used for preparing buffers that may be used as the first buffer in a packet must contain at least the number of elements specified in the scatterlist_size configuration field.
- If the DMA_HAS_EPIB or DMA_HAS_PSINFO flag bits are set, the first one or two entries in the scatterlist array must be filled in with info about these buffers. The buffers themselves must be writable and persistently allocated, as they will be written to before the completion function is called.
- Bits 27:24 of the flags argument to dmaengine_prep_slave_sg() function must contain the queue number passed in as parameter 2.

The receive code must call dma_rxfree_refill() to replenish the free buffer pools as needed.
This probably is best done at the end of the deferred work routine that also calls dma_poll(), but this depends on the circumstances.
When a receive channel is closed, and in certain exception situations, the rxpool_destructor function will be called to release the buffer.
This function is not called when a packet is returned through a completion queue.
The rxpool_destructor must unmap the buffer using dma_unmap_single(), dma_unmap_sg(), or similar functions. The buffer should then be freed.

Packet Reception Completion

When the PktDMA hardware has assembled a packet it places the address of the packet descriptor for the first buffer in the completion queue.
This may trigger an interrupt, causing the notification routine to be called, which may schedule deferred work.
The deferred work routine or other mechanism must then call the dma_poll() function, which will dequeue packet descriptors
from the completion queue and invoke the per-packet completion functions.
Note that in the case of a packet formed from a chain of buffers, only the completion routine for the first buffer in the chain will be invoked.

The per-packet completion function is passed a parameter whose value was set when the packet was allocated.
This value is not used by the PktDMA subsystem; however, it is the only mechanism for the completion function to find the received packet.
The Ethernet driver passes the address of a per-packet data structure that contains the scatterlist array, the “cookie”, and other critical data.

The completion function must perform several operations:

- Check the completion status of the packet using dma_async_is_tx_complete().
- Process the content of the EPIB and/or PSinfo sections.
- Unmap the buffer or chain of buffers. If a chain of buffers has been returned, the scatterlist array for the first buffer in the chain will be populated with the addresses and sizes of all the buffers in the chain.
- Free the buffer or buffers, or pass them to another subsystem that will.

DMA Channel Teardown

To tear down a PktDMA channel, bring the channel to a quiescent state, then call the dma_release_channel() function.
Any packets queued for transmission in a TX queue will be flushed by passing them to the completion routine with an error status.
Buffers queued to the RX free buffer pools will be passed to the rxpool_destructor function for disposal.

Packet Accelerator

- The Packet Accelerator driver is at **drivers/net/ethernet/ti/keystone_pa.c**.
  - The Packet Accelerator has 6 PDSPs.
  - The packet classifier image 1 is downloaded to PDSP0, PDSP1 and PDSP2.
  - The packet classifier image 2 is downloaded to PDSP3.
  - The packet modifier image is downloaded to PDSP4 and PDSP5.

PA Timestamping

- PA timestamping has been implemented in the network driver.
- All Receive packets will be timestamped and this timestamped by PDSP0 and this timestamp will be available in the timestamp field of the descriptor itself.
- To obtain the TX timestamp, we call a PA API to format the TX packet. Essentially what we do is to add a set of params to the "PSDATA" section of the descriptor. This packet is then sent to PDSP5. Internally this will route the packet to the switch. The TX timestamp is obtained in a return packet and we have designed the network driver in such a way to obtain both ethernet RX packet and the TX timestamp packet on the same queue and flow.
- The way we differentiate between an Ethernet RX packet and a TX timestmap packet is based on the "Software info 0" field of the descriptor.
- To obtain the timestamps itself, we use generic kernel APIs and features.
- Appropriate documentation for this can be found at Timestamping Documentation (http://www.mjmwired.net/kernel/Documentation/networking/timestamping.txt)

- The timestamping was tested with open source timestamping test code found at Timestamping Test Code (http://git.kernel.org/?p=linux/kernel/git/stable/linux-stable.git;a=tree;f=Documentation/networking/timestamping;h=9756a1d6b7fa1945c452e52b14c13cd0ca575090;hb=805a6af8dba5dfdd35ec35dc52ec0122400b2610)

```
./timestamping eth0 SOF_TIMESTAMPING_TX_HARDWARE SOF_TIMESTAMPING_SYS_HARDWARE SOF_TIMESTAMPING_RAW_HARDWARE
```

- Please refer to this (http://processors.wiki.ti.com/images/c/cb/Testing_timestamp_using_timestamping_app.pdf) document for testing PA timestamps in MCSDK3.0

Mark_mcast_match Special Packet Processing Feature

This feature provide for special packet egress processing for specific marked packets. The intended use is:

```
1) SOC Configured in multiple-interface mode
2) CPSW ALE re-enabled via /sys/class/net/eth0/device/ale_control (so that SOC switch is
   active behind the scenes)
3) NetCP interfaces slaved to a bridge
4) NetCP interfaces feed a common QoS tree
5) Bridge forwarding disabled via "ebtables -P FORWARD DROP" (because CPSW is
   doing the port to port forwarding)
```

In this rather odd situation, the bridge will transmit locally generated multicast (and broadcast) packets by sending one on each of the slaved interfaces (i.e. bridge flooding). This has two ramifications:

```
 (a) This results in multiple packets (copies of these locally generated
     muliticasts) through a common QoS, which is considered "bad"
     because the common QOS tree is configured assuming only one copy.
 (b) even if QOS is not present, sending multiple copies of these multicasts is
     sub-optimal since the CPSW switch is capable of doing the forwarding itself given
     just one copy of the original packet.
```

To avoid these ramifications, such local multicast packets can be marked via ebtables for special processing in the NetCP PA module before the packets are queued for transmission. Packets thus recognized are NOT marked for egress via a specific slave port, and thus will be transmitted through all slave ports by the CPSW h/w forwarding logic.

To do this, a new DTS parameter "mark_mcast_match" has been added. This parameter takes two u32 values: a "match" value and a "mask" value.

When the NetCP PA module encounters a packet with a non-zero skb->mark field, it bitwise-ANDs the skb->mark value with the "mask" value and then compares the result with the "match" value. If these do not match, the mark is ignored and the packet is processed normally.

However, if the "match" value matches, then the low-order 8 bits of the skb->mark field is used as a bitmask to determine whether the packet should be dropped. If the packet would normally have been directed to slave port 1, then bit 0 of skb->mark is checked; slave port 2 checks bit 1, etc. If the bit is set, then the packet is enqueued for ALE processing but *with the CPSW engress port field in the descriptor set to 0* (indicating that CPSW is responsible for selecting the egress port(s) to forward the packet too) ; if the bit is NOT set, the packet is silently dropped.

An example...

The device tree contains this PA definition:

mark_mcast_match = <0x12345a00 0xffffff00>;

The runtime configuration scripts execute this command:

ebtables -A OUTPUT -d Multicast -j mark \ --mark-set 0x12345a01 --mark-target ACCEPT

When the bridge attempts to send an ARP (broadcast) packet, it will send one packet to each of the slave interfaces. The packet sent by the bridge to slave interface eth0 (CPSW slave port 1) will be passed to the CPSW, and the ALE will broadcast this packet on all slave ports. The packets sent by the bridge to other slave interfaces (eth1, CPSW slave port 2) will be silently dropped.

SGMII

- The SGMII driver code is at **drivers/net/ethernet/ti/keystone_sgmii.c**.

The SGMII module on Keystone 2 devices can be configured to operate in various modes.
The modes are as follows

- mac mac autonegotiate
- mac phy
- mac mac forced
- mac fiber

The mode of operation can be decided through the device tree bindings. An example is shown below

```
slaves {
        slave0 {
                label           = "slave0";
                link-interface  = <1>;
                phy-handle = <&phy0>;
        };
        slave1 {
                label           = "slave1";
                link-interface  = <1>;
                phy-handle = <&phy1>;
        };

};
```

AS we can see in the above diagrarm, the **link-interface** attribute must be appropriately changed to decide the mode of operation.

- For mac mac auto negotiate use 0

- For mac phy use 1
- For mac mac forced use 2
- For mac fiber use 3

Note: 66AK2E supports 8 Ethernet (SGMII) ports, 2 ports to the EVM PHYs, 2 ports to AMC connector, and 4 ports to RTM connector. When using Mistral RTM BoC for those 4 PHYs on it, the SGMII ports are connected in reverse order. Instead of SGMII4 connected to PHY0 on the RTM BoC, it is connected to PHY3. Therefore, the device tree for the Mistral RTM BoC looks like:

```
mdio: mdio@24200f00 {
        phy4: phy@4 {
                compatible = "marvell,88e1145";
                reg = <4>;
        };
        phy5: phy@5 {
                compatible = "marvell,88e1145";
                reg = <5>;
        };
        phy6: phy@6 {
                compatible = "marvell,88e1145";
                reg = <6>;
        };
        phy7: phy@7 {
                compatible = "marvell,88e1145";
                reg = <7>;
        };
};
netcp {
        cpsw: cpsw@24200000 {
                slaves {
                        slave4 {
                                label = "slave4";
                                link-interface = <1>;
                                phy-handle = <&phy7>;
                        };
                        slave5 {
                                label = "slave5";
                                link-interface = <1>;
                                phy-handle = <&phy6>;
                        };
                        slave6 {
                                label = "slave6";
                                link-interface = <1>;
                                phy-handle = <&phy5>;
                        };
                        slave7 {
                                label = "slave7";
                                link-interface = <1>;
                                phy-handle = <&phy4>;
                        };
                };
        };
};
```

Setting up an NFS filesystem

An NFS filesystem can be setup during development.

To setup an NFS filesystem, first obtain a tarball of the filesystem. This should be untarred on the linux host machine.
This section will explain how to setup an NFS server on a Ubuntu linux host.

- Untar the filesystem that is made as part of the release into location **/opt/filesys**
- On the linux host open file **/etc/exports** and add the following line

```
/opt/filesys *(rw,subtree_check,no_root_squash,no_all_squash,sync)
```

- Please note that the location of the fielsystem on the host should be the same as that in /etc/exports
- Next we have to start the nfs server and this is achieved by giving the following command

```
/etc/init.d/nfs-kernel-server restart
```

Modifying the command line for NFS filesystem boot

The kernel command line should be appropriately modified to enable kernel boot up with an NFS filesystem

Add the following to the kernel command line **root=/dev/nfs nfsroot=192.168.1.140:/opt/filesys,v3,tcp rw ip=dhcp**

To use eth1 set ***ip=dhcp:eth1***

During kernel boot up, in the boot up log we should be able to see the following

Kernel command line: earlyprintk debug console=ttyS0,115200n8 ip=dhcp mem=512M rootwait=1 rootfstype=nfs root=/dev/nfs rw nfsroot=192.168.1.140:/opt/filesys,v3,tcp

Modifying CPSW Configurations Through Sysfs User Interface

Through sysfs, an user can show or modify some ALE control, ALE table and CPSW control configurations from user space by using the commands described in the following sub-sections.

1. Showing ALE Table

- Command to show the table entries.

```
$ cat /sys/devices/soc.0/2090000.netcp/ale_table
```

- One execution of the command may show only part of the table. Consecutive executions of the command will show the remaining parts of the table (see example below).
- The '+' sign at the end of the show indicates that there are entries in the remaining table not shown in the current execution of the command (see example below).

2. Showing RAW ALE Table

- Command to show the raw table entries.

```
$ cat /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

- Command to set the start-showing-index to n.

```
$ echo n > /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

- Only raw entries (without interpretation) will be shown.
- Depending on the number of occupied entries, it is more likely to show the whole table with one execution of the raw table show command. If not, consecutive executions of the command will show the remaining parts of the table.
- The '+' sign at the end of the show indicates that there are entries in the remaining table not shown in the current execution of the command (see example below).

3. Showing ALE Controls

- Command to show the ale controls.

```
$ cat /sys/devices/soc.0/2090000.netcp/ale_control
```

4. Showing CPSW Controls

- Command to show various CPSW controls

```
$ cat /sys/devices/soc.0/2090000.netcp/cpsw/file_name
```

where file_name is a file under the directory /sys/devices/soc.0/2090000.netcp/cpsw.

- Files or directories under the cpsw directory are
    - control
    - flow_control
    - port_tx_pri_map/
    - port_vlan/
    - priority_type
    - version
- For example, to see the CPSW version, use the command

```
$ cat /sys/devices/soc.0/2090000.netcp/cpsw/version
```

5. Adding/Deleting ALE Table Entries

- In general, the ALE Table add command is of the form

```
$ echo "add_command_format" > /sys/devices/soc.0/2090000.netcp/ale_table
```

or

```
$ echo "add_command_format" > /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

- The delete command is of the form

```
$ echo "n:" > /sys/devices/soc.0/2090000.netcp/ale_table
```

or

```
$ echo "n:" > /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

where n is the index of the table entry to be deleted.

- Command Formats
    - Adding VLAN command format

```
v.vid=(int).force_untag_egress=(hex 3b).reg_fld_mask=(hex 3b).unreg_fld_mask=(hex 3b).mem_list=(hex 3b)
```

    - Adding OUI Address command format

```
o.addr=(aa:bb:cc)
```

    - Adding Unicast Address command format

```
u.port=(int).block=(1|0).secure=(1|0).ageable=(1|0).addr=(aa:bb:cc:dd:ee:ff)
```

    - Adding Multicast Address command format

```
m.port_mask=(hex 3b).supervisory=(1|0).mc_fw_st=(int 0|1|2|3).addr=(aa:bb:cc:dd:ee:ff)
```

    - Adding VLAN Unicast Address command format

```
vu.port=(int).block=(1|0).secure=(1|0).ageable=(1|0).addr=(aa:bb:cc:dd:ee:ff).vid=(int)
```

    - Adding VLAN Multicast Address command format

```
vm.port_mask=(hex 3b).supervisory=(1|0).mc_fw_st=(int 0|1|2|3).addr=(aa:bb:cc:dd:ee:ff).vid=(int)
```

    - Deleting ALE Table Entry

```
entry_index:
```

    - Remark: any field that is not specified defaults to 0, except vid which defaults to -1 (i.e. no vid).
- Examples
    - Add a VLAN with vid=100 reg_fld_mask=0x7 unreg_fld_mask=0x2 mem_list=0x4

```
$ echo "v.vid=100.reg_fld_mask=0x7.unreg_fld_mask=0x2.mem_list=0x4" > /sys/class/net/eth0/device/ale_table
```

    - Add a persistent unicast address 02:18:31:7E:3E:6F

```
$ echo "u.addr=02:18:31:7E:3E:6F" > /sys/class/net/eth0/device/ale_table
```

    - Delete the 100-th entry in the table

```
$ echo "100:"  > /sys/class/net/eth0/device/ale_table
```

5. Modifying ALE Controls

- Access to the ALE Controls is available through the /sys/class/net/eth0/device/ale_control pseudo file. This file contains the following:
- • version: the ALE version information
- • **enable**: 0 to disable the ALE, 1 to enable ALE (should be 1 for normal operations)
- • **clear**: set to 1 to clear the table (refer to [1] for description)
- • **ageout** : set to 1 to force age out of entries (refer to [1] for description])
- • **p0_uni_flood_en** : set to 1 to enable unknown unicasts to be flooded to host port. Set to 0 to not flood such unicasts. Note: if set to 0, CPSW may delay sending packets to the SOC host until it learns what mac addresses the host is using.
- • **vlan_nolearn** : set to 1 to prevent VLAN id from being learned along with source address.
- • **no_port_vlan** : set to 1 to allow processing of packets received with VLAN ID=0; set to 0 to replace received packets with VLAN ID=0 to the VLAN set in the port's default VLAN register.
- • **oui_deny** : 0/1 (refer to [1] for a description of this bit)
- • **bypass**: set to 1 to enable ALE bypass. In this mode the CPSW will not act as switch on receive; instead it will forward all received traffic from external ports to the host port. Set to 0 for normal (switched) operations.
- • **rate_limit_tx**: set to 1 for rate limiting to apply to transmit direction, set to 0 for receive direction. Refer to [1] for a description of this bit.
- • **vlan_aware**: set to 1 to force the ALE into VLAN aware mode

- • **auth_enable**: set to 1 to enable table update by host only. Refer to [1] for more details on this feature
- • **rate_limit**: set to 1 to enable multicast/broadcast rate limiting feature. Refer to [1] for more details.
- • **port_state.0=** set the port 0 (host port) state. State can be:
- o 0: disabled
- o 1: blocked
- o 2: learning
- o 3: forwarding
- • **port_state.1**: set the port 1 state.
- • **port_state.2**: set the port 2 state
- • **drop_untagged.0** : set to 1 to drop untagged packets received on port 0 (host port)
- • **drop_untagged.1** : set to 1 to drop untagged packets received on port 1
- • **drop_untagged.2** : set to 1 to drop untagged packets received on port 2
- • **drop_unknown.0** : set to 1 to drop packets received on port 0 (host port) with unknown VLAN tags. Set to 0 to allows these to be processed
- • **drop_unknown.1** : set to 1 to drop packets received on port 1 with unknown VLAN tags. Set to 0 to allow these to be processed.
- • **drop_unknown.2** : set to 1 to drop packets received on port 2 with unknown VLAN tags. Set to 0 to allow these to be processed.
- • **nolearn.0** : set to 1 to disable address learning for port 0
- • **nolearn.1** : set to 1 to disable address learning for port 1
- • **nolearn.2** : set to 1 to disable address learning for port 2
- • **unknown_vlan_member** : this is the port mask for packets received with unknown VLAN IDs. The port mask is a 5 bit number with a bit representing each port. Bit 0 refers to the host port. A '1' in bit position N means include the port in further forwarding decision. (e.g., port mask = 0x7 means ports 0 (internal), 1 and 2 should be included in the forwarding decision). Refer to [1] for more details.
- • **unknown_mcast_flood=** : this is the port mask for packets received with unkwown VLAN ID and unknown (un-registered) destination multicast address. This port_mask will be used in the multicast flooding decision. unknown multicast flooding.
- • **unknown_reg_flood**: this is the port mask for packets received with unknown VLAN ID and registered (known) destination multicast address. It is used in the multicast forwarding decision.
- • **unknown_force_untag_egress**: this is a port mask to control if VLAN tags are stripped off on egress or not. Set to 1 to force tags to be stripped by h/w prior to transmission
- • **bcast_limit.0** : threshold for broadcast pacing on port 0 .
- • **bcast_limit.1**: threshold for broadcast pacing on port 1.
- • **bcast_limit.2** : threshold for broadcast pacing on port 2 .
- • **mcast_limit.0**: threshold for multicast pacing on port 0 .
- • **mcast_limit.1**: threshold for multicast pacing on port 1 ..
- • **mcast_limit.2**: threshold for multicast pacing on port 2 .
- Command format for each modifiable ALE control is the same as what is displayed for that field from showing the ALE table.
- For example, to disable ALE learning on port 0, use the command

```
$ echo "nolearn.0=0" > cat /sys/devices/soc.0/2090000.netcp/ale_control
```

6. Modifying CPSW Controls

- Command format for each modifiable CPSW control is the same as what is displayed for that field from showing the CPSW controls.
- For example, to enable flow control on port 2, use the command

```
$ echo "port2_flow_control_en=1" > /sys/devices/soc.0/2090000.netcp/cpsw/flow_control
```

7. Resetting CPSW Statistics

- Use the command

```
$ echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/A
```

or

```
$ echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/B
```

to reset statistics module A or B counters.

8. Additional Examples

To enable CPSW:

//enable unknown unicast flood to host, disable bypass, enable VID=0 processing
echo "po_uni_flood_en=1" > /sys/class/net/eth0/device/ale_control
echo "bypass=0" > /sys/class/net/eth0/device/ale_control
echo "no_port_vlan=1" > /sys/class/net/eth0/device/ale_control

To disable CPSW:
// disable port 0 flood for unknown unicast;
//enable bypass mode
echo "po_uni_flood_en=0" > /sys/class/net/eth0/device/ale_control
echo "bypass=1" > /sys/class/net/eth0/device/ale_control

To set port 1 state to forwarding:
echo "port_state.1=3" > /sys/class/net/eth0/device/ale_control

To set CPSW to VLAN aware mode:
echo "vlan_aware=1" > /sys/class/net/eth0/device/cpsw/control
echo "ale_vlan_aware=1" > /sys/class/net/eth0/device/ale_control
(set these to 0 to disable vlan aware mode)

To set port 1's Ingress VLAN defaults:
echo "port.1._vlan_id=5" > /sys/class/net/eth0/device/cpsw/port_vlan/1

echo "port.1._cfi=0" > /sys/class/net/eth0/device/cpsw/port_vlan/1
echo "port.1._vlan_pri=0" > /sys/class/net/eth0/device/cpsw/port_vlan/1

To set port 1 to use the above default vlan id on ingress:
echo "port.1.pass_pri_tagged=0" > /sys/class/net/eth0/device/cpsw/control

To set port 1's Egress VLAN defaults:
• For registered VLANs, the egress policy is set in the "force_untag_egress field" of the ALE entry for that VLAN. This field is a bit map with one bit per port. Port 0 is the host port. For example, to set VLAN #100 to force untagged egress on port 2 only:
echo "v.vid=100.force_untag_egress=0x4.reg_fld_mask=0x7.unreg_fld_mask=0x2.mem_list=0x4" > /sys/class/net/eth0/device/ale_table
• For un-registered VLANs, the egress policy is set in the ALE unknown vlan register, which is accessed via the ale_control pseudo file. The value is a bit map, one bit per port (port 0 is the host port). for example, set every port to drop unknown VLAN tags on egress
echo "unknown_force_untag_egress=7" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Admit tagged" (i.e. drop un-tagged) :
echo "drop_untagged.1=1" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Admit all" :
echo "drop_untagged.1=0" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Admit unknown VLAN":
echo "drop_unknown.1=0" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Drop unknown VLAN":
echo "drop_unknown.1=1" > /sys/class/net/eth0/device/ale_control

9. Sample Display

```
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/devices/soc.0/2090000.netcp/
-rw-r--r--    1 root     root          4096 Feb 28 20:32 ale_control
-r--r--r--    1 root     root          4096 Feb 28 20:30 ale_table
drwxr-xr-x    2 root     root             0 Feb 28 20:46 cpsw
lrwxrwxrwx    1 root     root             0 Jan  1  1970 driver -> ../../../bus/platform/drivers/keystone-netcp
-r--r--r--    1 root     root          4096 Feb 28 20:46 modalias
drwxr-xr-x    4 root     root             0 Jan  1  1970 net
drwxr-xr-x    2 root     root             0 Feb 28 20:46 power
lrwxrwxrwx    1 root     root             0 Jan  1  1970 subsystem -> ../../../bus/platform
-rw-r--r--    1 root     root          4096 Jan  1  1970 uevent
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/devices/soc.0/2090000.netcp/cpsw
-rw-r--r--    1 root     root          4096 Feb 28 20:31 control
-rw-r--r--    1 root     root          4096 Feb 28 20:31 flow_control
drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_tx_pri_map
drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_vlan
-rw-r--r--    1 root     root          4096 Feb 28 20:31 priority_type
drwxr-xr-x    2 root     root             0 Feb 28 20:31 stats
-r--r--r--    1 root     root          4096 Feb 28 20:31 version
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/class/net/eth0/device/
-rw-r--r--    1 root     root          4096 Feb 28 20:32 ale_control
-r--r--r--    1 root     root          4096 Feb 28 20:32 ale_table
drwxr-xr-x    2 root     root             0 Feb 28 20:30 cpsw
lrwxrwxrwx    1 root     root             0 Jan  1  1970 driver -> ../../../bus/platform/drivers/keystone-netcp
-r--r--r--    1 root     root          4096 Feb 28 20:32 modalias
drwxr-xr-x    4 root     root             0 Jan  1  1970 net
drwxr-xr-x    2 root     root             0 Feb 28 20:32 power
lrwxrwxrwx    1 root     root             0 Jan  1  1970 subsystem -> ../../../bus/platform
-rw-r--r--    1 root     root          4096 Jan  1  1970 uevent
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/class/net/eth0/device/cpsw/
-rw-r--r--    1 root     root          4096 Feb 28 20:31 control
-rw-r--r--    1 root     root          4096 Feb 28 20:31 flow_control
drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_tx_pri_map
drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_vlan
-rw-r--r--    1 root     root          4096 Feb 28 20:31 priority_type
drwxr-xr-x    2 root     root             0 Feb 28 20:31 stats
-r--r--r--    1 root     root          4096 Feb 28 20:31 version
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/version
cpsw version 1.3 (1) SGMII identification value 0x4ed1
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/control
fifo_loopback=0
vlan_aware=1
p0_enable=1
p0_pass_pri_tagged=0
p1_pass_pri_tagged=0
p2_pass_pri_tagged=0
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/flow_control
port0_flow_control_en=1
port1_flow_control_en=0
port2_flow_control_en=0
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/priority_type
escalate_pri_load_val=0
port0_pri_type_escalate=0
port1_pri_type_escalate=0
port2_pri_type_escalate=0
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/class/net/eth0/device/cpsw/port_tx_pri_map/
-rw-r--r--    1 root     root          4096 Feb 28 21:06 1
-rw-r--r--    1 root     root          4096 Feb 28 21:06 2
-rw-r--r--    1 root     root          4096 Feb 28 21:06 3
-rw-r--r--    1 root     root          4096 Feb 28 21:06 4
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/1
port_tx_pri_0=1
port_tx_pri_1=0
port_tx_pri_2=0
```

```
port_tx_pri_3=1
port_tx_pri_4=2
port_tx_pri_5=2
port_tx_pri_6=3
port_tx_pri_7=3
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/2
port_tx_pri_0=1
port_tx_pri_1=0
port_tx_pri_2=0
port_tx_pri_3=1
port_tx_pri_4=2
port_tx_pri_5=2
port_tx_pri_6=3
port_tx_pri_7=3
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/3
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/4
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/class/net/eth0/device/cpsw/port_vlan/
-rw-r--r--    1 root     root          4096 Feb 28 20:30 0
-rw-r--r--    1 root     root          4096 Feb 28 20:30 1
-rw-r--r--    1 root     root          4096 Feb 28 20:30 2
-rw-r--r--    1 root     root          4096 Feb 28 20:30 3
-rw-r--r--    1 root     root          4096 Feb 28 20:30 4
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/0
port_vlan_id=0
port_cfi=0
port_vlan_pri=0
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/1
port_vlan_id=0
port_cfi=0
port_vlan_pri=0
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/2
port_vlan_id=0
port_cfi=0
port_vlan_pri=0
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/3
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/4
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_control
version=(ALE_ID=0x0029) Rev 1.3
enable=1
clear=0
ageout=0
p0_uni_flood_en=0
vlan_nolearn=0
no_port_vlan=1
oui_deny=0
bypass=1
rate_limit_tx=0
vlan_aware=1
auth_enable=0
rate_limit=0
port_state.0=3
port_state.1=3
port_state.2=0
drop_untagged.0=0
drop_untagged.1=0
drop_untagged.2=0
drop_unknown.0=0
drop_unknown.1=0
drop_unknown.2=0
nolearn.0=0
nolearn.1=0
nolearn.2=0
unknown_vlan_member=7
unknown_mcast_flood=6
unknown_reg_flood=7
unknown_force_untag_egress=7
bcast_limit.0=0
bcast_limit.1=0
bcast_limit.2=0
mcast_limit.0=0
mcast_limit.1=0
mcast_limit.2=0
root@keystone-evm:~# echo "p0_uni_flood_en=1" >  /sys/class/net/eth0/device/ale_control
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_control | grep p0_uni_flood_en
p0_uni_flood_en=1
root@keystone-evm:~#
root@keystone-evm:~# echo "drop_unknown.1=1" > /sys/class/net/eth0/device/ale_control
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_control | grep drop_unknown
drop_unknown.0=0
drop_unknown.1=1
drop_unknown.2=0
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table
index 0, raw: 0000001c d000ffff ffffffff, type: addr(1), addr: ff:ff:ff:ff:ff:ff, mcstate: f(3), port mask: 7, no super
index 1, raw: 00000000 10000017 eaf4323a, type: addr(1), addr: 00:17:ea:f4:32:3a, uctype: persistant(0), port: 0
index 2, raw: 0000001c d0003333 00000001, type: addr(1), addr: 33:33:00:00:00:01, mcstate: f(3), port mask: 7, no super
index 3, raw: 0000001c d0000100 5e000001, type: addr(1), addr: 01:00:5e:00:00:01, mcstate: f(3), port mask: 7, no super
index 4, raw: 00000004 f0000001 297495bf, type: vlan+addr(3), addr: 00:01:29:74:95:bf, vlan: 0, uctype: touched(3), port: 1
index 5, raw: 0000001c d0003333 fff4323a, type: addr(1), addr: 33:33:ff:f4:32:3a, mcstate: f(3), port mask: 7, no super
index 6, raw: 00000004 f0000000 0c07acca, type: vlan+addr(3), addr: 00:00:0c:07:ac:ca, vlan: 0, uctype: touched(3), port: 1
index 7, raw: 00000004 7000e8e0 b75db25e, type: vlan+addr(3), addr: e8:e0:b7:5d:b2:5e, vlan: 0, uctype: untouched(1), port: 1
index 9, raw: 00000004 f0005c26 0a69440b, type: vlan+addr(3), addr: 5c:26:0a:69:44:0b, vlan: 0, uctype: touched(3), port: 1
index 11, raw: 00000004 70005c26 0a5b2ea6, type: vlan+addr(3), addr: 5c:26:0a:5b:2e:a6, vlan: 0, uctype: untouched(1), port: 1
index 12, raw: 00000004 f000d4be d93db6b8, type: vlan+addr(3), addr: d4:be:d9:3d:b6:b8, vlan: 0, uctype: touched(3), port: 1
index 13, raw: 00000004 70000014 225b62d9, type: vlan+addr(3), addr: 00:14:22:5b:62:d9, vlan: 0, uctype: untouched(1), port: 1
index 14, raw: 00000004 7000000b 7866c6d3, type: vlan+addr(3), addr: 00:0b:78:66:c6:d3, vlan: 0, uctype: untouched(1), port: 1
index 15, raw: 00000004 f0005c26 0a6952fa, type: vlan+addr(3), addr: 5c:26:0a:69:52:fa, vlan: 0, uctype: touched(3), port: 1
index 16, raw: 00000004 f000b8ac 6f7d1b65, type: vlan+addr(3), addr: b8:ac:6f:7d:1b:65, vlan: 0, uctype: touched(3), port: 1
index 17, raw: 00000004 7000d4be d9a34760, type: vlan+addr(3), addr: d4:be:d9:a3:47:60, vlan: 0, uctype: untouched(1), port: 1
index 18, raw: 00000004 70000007 eb645149, type: vlan+addr(3), addr: 00:07:eb:64:51:49, vlan: 0, uctype: untouched(1), port: 1
index 19, raw: 00000004 f3200000 0c07acd3, type: vlan+addr(3), addr: 00:00:0c:07:ac:d3, vlan: 800, uctype: touched(3), port: 1
index 20, raw: 00000004 7000d067 e5e7330c, type: vlan+addr(3), addr: d0:67:e5:e7:33:0c, vlan: 0, uctype: untouched(1), port: 1
index 22, raw: 00000004 70000026 b9802a50, type: vlan+addr(3), addr: 00:26:b9:80:2a:50, vlan: 0, uctype: untouched(1), port: 1
index 23, raw: 00000004 f000d067 e5e5aa12, type: vlan+addr(3), addr: d0:67:e5:e5:aa:12, vlan: 0, uctype: touched(3), port: 1
index 24, raw: 00000004 f0000011 430619f6, type: vlan+addr(3), addr: 00:11:43:06:19:f6, vlan: 0, uctype: touched(3), port: 1
index 25, raw: 00000004 7000bc30 5bde7ee2, type: vlan+addr(3), addr: bc:30:5b:de:7e:e2, vlan: 0, uctype: untouched(1), port: 1
```

```
index 26, raw: 00000004 7000b8ac 6f92c3d3, type: vlan+addr(3), addr: b8:ac:6f:92:c3:d3, vlan: 0, uctype: untouched(1), port: 1
index 28, raw: 00000004 f0000012 01f7d6ff, type: vlan+addr(3), addr: 00:12:01:f7:d6:ff, vlan: 0, uctype: touched(3), port: 1
index 29, raw: 00000004 f000000b db7789a5, type: vlan+addr(3), addr: 00:0b:db:77:89:a5, vlan: 0, uctype: touched(3), port: 1
index 31, raw: 00000004 70000018 8b2d9433, type: vlan+addr(3), addr: 00:18:8b:2d:94:33, vlan: 0, uctype: untouched(1), port: 1
index 32, raw: 00000004 70000013 728a0dc0, type: vlan+addr(3), addr: 00:13:72:8a:0d:c0, vlan: 0, uctype: untouched(1), port: 1
index 33, raw: 00000004 700000c0 b76f6e82, type: vlan+addr(3), addr: 00:c0:b7:6f:6e:82, vlan: 0, uctype: untouched(1), port: 1
index 34, raw: 00000004 700014da e9096f9a, type: vlan+addr(3), addr: 14:da:e9:09:6f:9a, vlan: 0, uctype: untouched(1), port: 1
index 35, raw: 00000004 f0000023 24086746, type: vlan+addr(3), addr: 00:23:24:08:67:46, vlan: 0, uctype: touched(3), port: 1
index 36, raw: 00000004 7000001b 11b4362f, type: vlan+addr(3), addr: 00:1b:11:b4:36:2f, vlan: 0, uctype: untouched(1), port: 1
[0..36]: 32 entries, +
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table
index 37, raw: 00000004 70000019 b9382f7e, type: vlan+addr(3), addr: 00:19:b9:38:2f:7e, vlan: 0, uctype: untouched(1), port: 1
index 38, raw: 00000004 f3200011 93ec6fa2, type: vlan+addr(3), addr: 00:11:93:ec:6f:a2, vlan: 800, uctype: touched(3), port: 1
index 40, raw: 00000004 f0000012 01f7a73f, type: vlan+addr(3), addr: 00:12:01:f7:a7:3f, vlan: 0, uctype: touched(3), port: 1
index 41, raw: 00000004 f0000011 855b1f3c, type: vlan+addr(3), addr: 00:11:85:5b:1f:3c, vlan: 0, uctype: touched(3), port: 1
index 42, raw: 00000004 7000d4be d900d37e, type: vlan+addr(3), addr: d4:be:d9:00:d3:7e, vlan: 0, uctype: untouched(1), port: 1
index 45, raw: 00000004 f3200012 01f7d6ff, type: vlan+addr(3), addr: 00:12:01:f7:d6:ff, vlan: 800, uctype: touched(3), port: 1
index 46, raw: 00000004 f0000002 fcc039df, type: vlan+addr(3), addr: 00:02:fc:c0:39:df, vlan: 0, uctype: touched(3), port: 1
index 47, raw: 00000004 f0000000 0c07ac66, type: vlan+addr(3), addr: 00:00:0c:07:ac:66, vlan: 0, uctype: touched(3), port: 1
index 48, raw: 00000004 f000d4be d94167da, type: vlan+addr(3), addr: d4:be:d9:41:67:da, vlan: 0, uctype: touched(3), port: 1
index 49, raw: 00000004 f000d067 e5e72bc0, type: vlan+addr(3), addr: d0:67:e5:e7:2b:c0, vlan: 0, uctype: touched(3), port: 1
index 50, raw: 00000004 f0005c26 0a6a51d0, type: vlan+addr(3), addr: 5c:26:0a:6a:51:d0, vlan: 0, uctype: touched(3), port: 1
index 51, raw: 00000004 70000014 22266425, type: vlan+addr(3), addr: 00:14:22:26:64:25, vlan: 0, uctype: untouched(1), port: 1
index 53, raw: 00000004 f3200002 fcc039df, type: vlan+addr(3), addr: 00:02:fc:c0:39:df, vlan: 800, uctype: touched(3), port: 1
index 54, raw: 00000004 f000000b cd413d26, type: vlan+addr(3), addr: 00:0b:cd:41:3d:26, vlan: 0, uctype: touched(3), port: 1
index 55, raw: 00000004 f3200000 0c07ac6f, type: vlan+addr(3), addr: 00:00:0c:07:ac:6f, vlan: 800, uctype: touched(3), port: 1
index 56, raw: 00000004 f000000b cd413d27, type: vlan+addr(3), addr: 00:0b:cd:41:3d:27, vlan: 0, uctype: touched(3), port: 1
index 57, raw: 00000004 f000000d 5620cdce, type: vlan+addr(3), addr: 00:0d:56:20:cd:ce, vlan: 0, uctype: touched(3), port: 1
index 58, raw: 00000004 f0000004 e2fceead, type: vlan+addr(3), addr: 00:04:e2:fc:ee:ad, vlan: 0, uctype: touched(3), port: 1
index 59, raw: 00000004 7000d4be d93db91b, type: vlan+addr(3), addr: d4:be:d9:3d:b9:1b, vlan: 0, uctype: untouched(1), port: 1
index 60, raw: 00000004 70000019 b9022455, type: vlan+addr(3), addr: 00:19:b9:02:24:55, vlan: 0, uctype: untouched(1), port: 1
index 61, raw: 00000004 f0000027 1369552b, type: vlan+addr(3), addr: 00:27:13:69:55:2b, vlan: 0, uctype: touched(3), port: 1
index 62, raw: 00000004 70005c26 0a06d1cd, type: vlan+addr(3), addr: 5c:26:0a:06:d1:cd, vlan: 0, uctype: untouched(1), port: 1
index 63, raw: 00000004 7000d4be d96816aa, type: vlan+addr(3), addr: d4:be:d9:68:16:aa, vlan: 0, uctype: untouched(1), port: 1
index 64, raw: 00000004 70000015 f28e329c, type: vlan+addr(3), addr: 00:15:f2:8e:32:9c, vlan: 0, uctype: untouched(1), port: 1
index 66, raw: 00000004 7000d067 e5e53caf, type: vlan+addr(3), addr: d0:67:e5:e5:3c:af, vlan: 0, uctype: untouched(1), port: 1
index 67, raw: 00000004 f000d4be d9416812, type: vlan+addr(3), addr: d4:be:d9:41:68:12, vlan: 0, uctype: touched(3), port: 1
index 69, raw: 00000004 f3200012 01f7a73f, type: vlan+addr(3), addr: 00:12:01:f7:a7:3f, vlan: 800, uctype: touched(3), port: 1
index 75, raw: 00000004 70000014 22266386, type: vlan+addr(3), addr: 00:14:22:26:63:86, vlan: 0, uctype: untouched(1), port: 1
index 80, raw: 00000004 70000030 6e5ee4b4, type: vlan+addr(3), addr: 00:30:6e:5e:e4:b4, vlan: 0, uctype: untouched(1), port: 1
index 83, raw: 00000004 70005c26 0a695379, type: vlan+addr(3), addr: 5c:26:0a:69:53:79, vlan: 0, uctype: untouched(1), port: 1
index 85, raw: 00000004 7000d4be d936b959, type: vlan+addr(3), addr: d4:be:d9:36:b9:59, vlan: 0, uctype: untouched(1), port: 1
index 86, raw: 00000004 7000bc30 5bde7ec2, type: vlan+addr(3), addr: bc:30:5b:de:7e:c2, vlan: 0, uctype: untouched(1), port: 1
[37..86]: 32 entries, +
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table
index 87, raw: 00000004 7000b8ac 6f7f4712, type: vlan+addr(3), addr: b8:ac:6f:7f:47:12, vlan: 0, uctype: untouched(1), port: 1
index 88, raw: 00000004 f0005c26 0a694420, type: vlan+addr(3), addr: 5c:26:0a:69:44:20, vlan: 0, uctype: touched(3), port: 1
index 89, raw: 00000004 f0000018 8b2d92e2, type: vlan+addr(3), addr: 00:18:8b:2d:92:e2, vlan: 0, uctype: touched(3), port: 1
index 93, raw: 00000004 7000001a a0a0c9df, type: vlan+addr(3), addr: 00:1a:a0:a0:c9:df, vlan: 0, uctype: untouched(1), port: 1
index 94, raw: 00000004 f000e8e0 b736b25e, type: vlan+addr(3), addr: e8:e0:b7:36:b2:5e, vlan: 0, uctype: touched(3), port: 1
index 96, raw: 00000004 70000010 18af5bfb, type: vlan+addr(3), addr: 00:10:18:af:5b:fb, vlan: 0, uctype: untouched(1), port: 1
index 99, raw: 00000004 70003085 a9a63965, type: vlan+addr(3), addr: 30:85:a9:a6:39:65, vlan: 0, uctype: untouched(1), port: 1
index 101, raw: 00000004 70005c26 0a695312, type: vlan+addr(3), addr: 5c:26:0a:69:53:12, vlan: 0, uctype: untouched(1), port: 1
index 104, raw: 00000004 7000f46d 04e22fc9, type: vlan+addr(3), addr: f4:6d:04:e2:2f:c9, vlan: 0, uctype: untouched(1), port: 1
index 105, raw: 00000004 7000001b 788de114, type: vlan+addr(3), addr: 00:1b:78:8d:e1:14, vlan: 0, uctype: untouched(1), port: 1
index 109, raw: 00000004 7000d4be d96816f4, type: vlan+addr(3), addr: d4:be:d9:68:16:f4, vlan: 0, uctype: untouched(1), port: 1
index 111, raw: 00000004 f0000010 18a113b5, type: vlan+addr(3), addr: 00:10:18:a1:13:b5, vlan: 0, uctype: touched(3), port: 1
index 115, raw: 00000004 f000f46d 04e22fbd, type: vlan+addr(3), addr: f4:6d:04:e2:2f:bd, vlan: 0, uctype: touched(3), port: 1
index 116, raw: 00000004 7000b8ac 6f8ed5e6, type: vlan+addr(3), addr: b8:ac:6f:8e:d5:e6, vlan: 0, uctype: untouched(1), port: 1
index 118, raw: 00000004 7000001a a0b2ebee, type: vlan+addr(3), addr: 00:1a:a0:b2:eb:ee, vlan: 0, uctype: untouched(1), port: 1
index 119, raw: 00000004 7000782b cbab87d4, type: vlan+addr(3), addr: 78:2b:cb:ab:87:d4, vlan: 0, uctype: untouched(1), port: 1
index 126, raw: 00000004 70000018 8b09703d, type: vlan+addr(3), addr: 00:18:8b:09:70:3d, vlan: 0, uctype: untouched(1), port: 1
index 129, raw: 00000004 70000050 b65f189e, type: vlan+addr(3), addr: 00:50:b6:5f:18:9e, vlan: 0, uctype: untouched(1), port: 1
index 131, raw: 00000004 f000bc30 5bd07ed1, type: vlan+addr(3), addr: bc:30:5b:d0:7e:d1, vlan: 0, uctype: touched(3), port: 1
index 133, raw: 00000004 f0003085 a9a26425, type: vlan+addr(3), addr: 30:85:a9:a2:64:25, vlan: 0, uctype: touched(3), port: 1
index 147, raw: 00000004 f000b8ac 6f8bae7f, type: vlan+addr(3), addr: b8:ac:6f:8b:ae:7f, vlan: 0, uctype: touched(3), port: 1
index 175, raw: 00000004 700090e2 ba02c6e4, type: vlan+addr(3), addr: 90:e2:ba:02:c6:e4, vlan: 0, uctype: untouched(1), port: 1
index 186, raw: 00000004 70000013 728c27fd, type: vlan+addr(3), addr: 00:13:72:8c:27:fd, vlan: 0, uctype: untouched(1), port: 1
index 197, raw: 00000004 f0000012 3f716cb1, type: vlan+addr(3), addr: 00:12:3f:71:6c:b1, vlan: 0, uctype: touched(3), port: 1
index 249, raw: 00000004 7000e89d 877c862f, type: vlan+addr(3), addr: e8:9d:87:7c:86:2f, vlan: 0, uctype: untouched(1), port: 1
[87..1023]: 25 entries
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table_raw
0: 1c d000ffff ffffffff
1: 00 10000017 eaf4323a
2: 1c d0003333 00000001
3: 1c d0000100 5e000001
4: 04 f0000001 297495bf
5: 1c d0003333 fff4323a
6: 04 f0000000 0c07acca
7: 04 7000e8e0 b75db25e
9: 04 f0005c26 0a69440b
11: 04 70005c26 0a5b2ea6
12: 04 f000d4be d93db6b8
13: 04 f0000014 225b62d9
14: 04 7000000b 7866c6d3
15: 04 f0005c26 0a6952fa
16: 04 f000b8ac 6f7d1b65
17: 04 7000d4be d9a34760
18: 04 70000007 eb645149
19: 04 f3200000 0c07acd3
20: 04 7000d067 e5e7330c
22: 04 70000026 b9802a50
23: 04 f000d067 e5e5aa12
24: 04 f0000011 430619f6
25: 04 f000bc30 5bde7ee2
26: 04 f000b8ac 6f92c3d3
28: 04 f0000012 01f7d6ff
29: 04 f000000b db7789a5
31: 04 70000018 8b2d9433
32: 04 70000013 728a0dc0
33: 04 700000c0 b76f6e82
34: 04 700014da e9096f9a
35: 04 f0000023 24086746
36: 04 7000001b 11b4362f
37: 04 f0000019 b9382f7e
38: 04 f3200011 93ec6fa2
39: 04 f0005046 5d74bf90
40: 04 f0000012 01f7a73f
41: 04 f0000011 855b1f3c
42: 04 f000d4be d900d37e
45: 04 f3200012 01f7d6ff
46: 04 f0000002 fcc039df
47: 04 f0000000 0c07ac66
48: 04 f000d4be d94167da
49: 04 f000d067 e5e72bc0
50: 04 f0005c26 0a6a51d0
51: 04 70000014 22266425
53: 04 f3200002 fcc039df
54: 04 f000000b cd413d26
```

```
55: 04 f3200000 0c07ac6f
56: 04 f000000b cd413d27
57: 04 f000000d 5620cdce
58: 04 f0000004 e2fceead
59: 04 7000d4be d93db91b
60: 04 70000019 b9022455
61: 04 f0000027 1369552b
62: 04 70005c26 0a06d1cd
63: 04 7000d4be d96816aa
64: 04 70000015 f28e329c
66: 04 7000d067 e5e53caf
67: 04 f000d4be d9416812
69: 04 f3200012 01f7a73f
75: 04 70000014 22266386
80: 04 70000030 6e5ee4b4
83: 04 70005c26 0a695379
85: 04 7000d4be d936b959
86: 04 7000bc30 5bde7ec2
87: 04 7000b8ac 6f7f4712
88: 04 f0005c26 0a694420
89: 04 f0000018 8b2d92e2
93: 04 7000001a a0a0c9df
94: 04 f000e8e0 b736b25e
96: 04 70000010 18af5bfb
99: 04 f0003085 a9a63965
101: 04 70005c26 0a695312
104: 04 7000f46d 04e22fc9
105: 04 7000001b 788de114
109: 04 7000d4be d96816f4
111: 04 f0000010 18a113b5
115: 04 f000f46d 04e22fbd
116: 04 7000b8ac 6f8ed5e6
118: 04 7000001a a0b2ebee
119: 04 7000782b cbab87d4
126: 04 70000018 8b09703d
129: 04 f0000050 b65f189e
131: 04 f000bc30 5bd07ed1
133: 04 f0003085 a9a26425
147: 04 f000b8ac 6f8bae7f
175: 04 700090e2 ba02c6e4
181: 04 f0000012 3f99c9dc
182: 04 f000000c f1d2df6b
186: 04 70000013 728c27fd
197: 04 f0000012 3f716cb1
249: 04 7000e89d 877c862f
[0..1023]: 92 entries
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/devices/soc.0/2090000.netcp/cpsw/stats
--w-------    1 root     root          4096 Feb 28 20:32 A
--w-------    1 root     root          4096 Feb 28 20:33 B
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# ethtool -S eth0
NIC statistics:
     CPSW_A:rx_good_frames: 133
     CPSW_A:rx_broadcast_frames: 0
     CPSW_A:rx_multicast_frames: 0
     CPSW_A:rx_pause_frames: 0
     CPSW_A:rx_crc_errors: 0
     CPSW_A:rx_align_code_errors: 0
     CPSW_A:rx_oversized_frames: 0
     CPSW_A:rx_jabber_frames: 0
     CPSW_A:rx_undersized_frames: 0
     CPSW_A:rx_fragments: 0
     CPSW_A:rx_bytes: 20878
     CPSW_A:tx_good_frames: 1469
     CPSW_A:tx_broadcast_frames: 553
     CPSW_A:tx_multicast_frames: 805
     CPSW_A:tx_pause_frames: 0
     CPSW_A:tx_deferred_frames: 0
     CPSW_A:tx_collision_frames: 0
     CPSW_A:tx_single_coll_frames: 0
     CPSW_A:tx_mult_coll_frames: 0
     CPSW_A:tx_excessive_collisions: 0
     CPSW_A:tx_late_collisions: 0
     CPSW_A:tx_underrun: 0
     CPSW_A:tx_carrier_sense_errors: 0
     CPSW_A:tx_bytes: 148064
     CPSW_A:tx_64byte_frames: 335
     CPSW_A:tx_65_to_127byte_frames: 834
     CPSW_A:tx_128_to_255byte_frames: 384
     CPSW_A:tx_256_to_511byte_frames: 45
     CPSW_A:tx_512_to_1023byte_frames: 4
     CPSW_A:tx_1024byte_frames: 0
     CPSW_A:net_bytes: 168942
     CPSW_A:rx_sof_overruns: 0
     CPSW_A:rx_mof_overruns: 0
     CPSW_A:rx_dma_overruns: 0
     CPSW_B:rx_good_frames: 923
     CPSW_B:rx_broadcast_frames: 355
     CPSW_B:rx_multicast_frames: 499
     CPSW_B:rx_pause_frames: 0
     CPSW_B:rx_crc_errors: 0
     CPSW_B:rx_align_code_errors: 0
     CPSW_B:rx_oversized_frames: 0
     CPSW_B:rx_jabber_frames: 0
     CPSW_B:rx_undersized_frames: 0
     CPSW_B:rx_fragments: 0
     CPSW_B:rx_bytes: 95208
     CPSW_B:tx_good_frames: 77
     CPSW_B:tx_broadcast_frames: 0
     CPSW_B:tx_multicast_frames: 0
     CPSW_B:tx_pause_frames: 0
     CPSW_B:tx_deferred_frames: 0
     CPSW_B:tx_collision_frames: 0
     CPSW_B:tx_single_coll_frames: 0
     CPSW_B:tx_mult_coll_frames: 0
     CPSW_B:tx_excessive_collisions: 0
     CPSW_B:tx_late_collisions: 0
     CPSW_B:tx_underrun: 0
     CPSW_B:tx_carrier_sense_errors: 0
     CPSW_B:tx_bytes: 12654
     CPSW_B:tx_64byte_frames: 201
     CPSW_B:tx_65_to_127byte_frames: 527
     CPSW_B:tx_128_to_255byte_frames: 236
     CPSW_B:tx_256_to_511byte_frames: 33
     CPSW_B:tx_512_to_1023byte_frames: 4
     CPSW_B:tx_1024byte_frames: 0
     CPSW_B:net_bytes: 107862
     CPSW_B:rx_sof_overruns: 0
```

```
     CPSW_B:rx_mof_overruns: 0
     CPSW_B:rx_dma_overruns: 0
root@keystone-evm:~# echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/A
root@keystone-evm:~#
root@keystone-evm:~# echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/B
root@keystone-evm:~#
root@keystone-evm:~# ethtool -S eth0
NIC statistics:
     CPSW_A:rx_good_frames: 5
     CPSW_A:rx_broadcast_frames: 0
     CPSW_A:rx_multicast_frames: 0
     CPSW_A:rx_pause_frames: 0
     CPSW_A:rx_crc_errors: 0
     CPSW_A:rx_align_code_errors: 0
     CPSW_A:rx_oversized_frames: 0
     CPSW_A:rx_jabber_frames: 0
     CPSW_A:rx_undersized_frames: 0
     CPSW_A:rx_fragments: 0
     CPSW_A:rx_bytes: 766
     CPSW_A:tx_good_frames: 5
     CPSW_A:tx_broadcast_frames: 0
     CPSW_A:tx_multicast_frames: 0
     CPSW_A:tx_pause_frames: 0
     CPSW_A:tx_deferred_frames: 0
     CPSW_A:tx_collision_frames: 0
     CPSW_A:tx_single_coll_frames: 0
     CPSW_A:tx_mult_coll_frames: 0
     CPSW_A:tx_excessive_collisions: 0
     CPSW_A:tx_late_collisions: 0
     CPSW_A:tx_underrun: 0
     CPSW_A:tx_carrier_sense_errors: 0
     CPSW_A:tx_bytes: 814
     CPSW_A:tx_64byte_frames: 0
     CPSW_A:tx_65_to_127byte_frames: 2
     CPSW_A:tx_128_to_255byte_frames: 8
     CPSW_A:tx_256_to_511byte_frames: 0
     CPSW_A:tx_512_to_1023byte_frames: 0
     CPSW_A:tx_1024byte_frames: 0
     CPSW_A:net_bytes: 1580
     CPSW_A:rx_sof_overruns: 0
     CPSW_A:rx_mof_overruns: 0
     CPSW_A:rx_dma_overruns: 0
     CPSW_B:rx_good_frames: 4
     CPSW_B:rx_broadcast_frames: 0
     CPSW_B:rx_multicast_frames: 1
     CPSW_B:rx_pause_frames: 0
     CPSW_B:rx_crc_errors: 0
     CPSW_B:rx_align_code_errors: 0
     CPSW_B:rx_oversized_frames: 0
     CPSW_B:rx_jabber_frames: 0
     CPSW_B:rx_undersized_frames: 0
     CPSW_B:rx_fragments: 0
     CPSW_B:rx_bytes: 814
     CPSW_B:tx_good_frames: 5
     CPSW_B:tx_broadcast_frames: 0
     CPSW_B:tx_multicast_frames: 0
     CPSW_B:tx_pause_frames: 0
     CPSW_B:tx_deferred_frames: 0
     CPSW_B:tx_collision_frames: 0
     CPSW_B:tx_single_coll_frames: 0
     CPSW_B:tx_mult_coll_frames: 0
     CPSW_B:tx_excessive_collisions: 0
     CPSW_B:tx_late_collisions: 0
     CPSW_B:tx_underrun: 0
     CPSW_B:tx_carrier_sense_errors: 0
     CPSW_B:tx_bytes: 766
     CPSW_B:tx_64byte_frames: 0
     CPSW_B:tx_65_to_127byte_frames: 2
     CPSW_B:tx_128_to_255byte_frames: 8
     CPSW_B:tx_256_to_511byte_frames: 0
     CPSW_B:tx_512_to_1023byte_frames: 0
     CPSW_B:tx_1024byte_frames: 0
     CPSW_B:net_bytes: 1580
     CPSW_B:rx_sof_overruns: 0
     CPSW_B:rx_mof_overruns: 0
     CPSW_B:rx_dma_overruns: 0
root@keystone-evm:~#
```

Using Ethtool to Change PHY Settings

Use the Linux "ethtool -s" command to change PHY settings of selected parameters as shown below

```
       ethtool -s|--change DEVNAME     Change generic options
              [ speed %d ]
              [ duplex half|full ]
              [ port tp|aui|bnc|mii|fibre ]
              [ autoneg on|off ]
              [ advertise %x ]
```

Enabling MDIO

In the default K2HK DTS, currently MDIO is disabled as a software workaround for the MDIO hardware issue on K2HK EVM. To enable MDIO driver on custom board, add the following bindings in the K2HK DTS for 1G. For u-boot has_mdio env variable should be set as well to enable MDIO hardware IP.

For K2HK EVM:

```
             mdio: mdio@2090300 {
                     compatible      = "ti,davinci_mdio";
                     #address-cells = <1>;
                     #size-cells = <0>;
                     reg             = <0x2090300 0x100>;
                     bus_freq        = <1000000>;
                     clocks          = <&clkcpgmac>;
                     clock-names     = "fck";
                     phy0: phy@0 {
                             compatible = "marvell,88e1111";
                             reg = <0>;
                     };
                     phy1: phy@1 {
```

```
                        compatible = "marvell,88e1111";
                        reg = <1>;
                };
        };
```

Also modify the slaveX nodes in the netcp node to the following:

```
    netcp {
            ...
            slaves {
                    slave0 {
                            label           = "slave0";
                            link-interface  = <1>;
                            phy-handle      = <&phy0>;
                    };
                    slave1 {
                            label           = "slave1";
                            link-interface  = <1>;
                            phy-handle      = <&phy1>;
                    };
            };
    };
```

Starting from MCSDK release 3.1.1, K2E and K2L MDIO are enabled by default. To enable MDIO on these platforms in previous MCSDK releases, see the k2e-net.dtsi or k2l-net.dtsi DTS file in MCSDK release 3.1.1.

Note: Enabling MDIO will have impact on the system bootup time due to the time spent in auto-negotiation on each ethernet port.

Common Platform Time Sync (CPTS)

The Common Platform Time Sync (CPTS) module is used to facilitate host control of time sync operations. It enables compliance with the IEEE 1588-2008 standard for a precision clock synchronization protocol.

Although CPTS timestamping co-exists with PA timestamping, CPTS timestamping is only for PTP packets and in that case, PA will not timestamp those packets.

CPTS Hardware Configurations

1. CPTS Device Tree Bindings Following are the CPTS related device tree bindings

- cpts_reg_ofs

  cpts register offset in cpsw module

- cpts_rftclk_sel

  chooses the input rftclk, default is 0

- cpts_rftclk_freq

  ref clock frequency in Hz if it is an **external** clock

- cpsw_cpts_rft_clk

  ref clock name if it is an **internal** clock

- cpts_ts_comp_length

  PPS Asserted Length (in Ref Clk Cycles)

- cpts_ts_comp_polarity

  if 1, PPS is assered high; otherwise asserted low

- cpts_clock_mult, cpts_clock_shift, cpts_clock_div

  multiplier and divider for converting cpts counter value to timestamp time

```
Example:
```

```
   netcp: netcp@2090000 {
      ...
      clocks = <&papllclk>, <&clkcpgmac>, <&chipclk12>;
      clock-names = "clk_pa", "clk_cpgmac", "cpsw_cpts_rft_clk";
      ...
      cpsw: cpsw@2090000 {
      ...
         cpts_reg_ofs = <0xd00>;
         ...
         cpts_rftclk_sel=<8>;
         /*cpts_rftclk_freq = <122800000>;*/
         cpts_ts_comp_length = <3>;
         cpts_ts_comp_polarity = <1>;  /* 1 - assert high */
         /* cpts_clock_mult = <6250>; */
         /* cpts_clock_shift = <8>; */
         /* cpts_clock_div = <3>; */
         ...
      };
      ...
   };
```

2. Configurations during driver initialization

By default, cpts is configured with the following configurations at boot up:

- Tx and Rx Annex D support but only one vlan tag (ts_vlan_ltype1_en)
- Tx and Rx Annex E support but only one vlan tag (ts_vlan_ltype1_en)
- Tx and Rx Annex F support but only one vlan tag (ts_vlan_ltype1_en)
- ts_vlan_ltype1 = 0x8100 (default)
- uni-cast enabled
- ttl_nonzero enabled

3. Configurations during runtime (Sysfs)

Currently the following sysfs are available for cpts related runtime configuration

- /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/n/uni_en

  (where n is slave port number)

  - Read/Write
  - 1 (enable unicast)
  - 0 (disable unicast)

- /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/n/mcast_addr

  (where n is slave port number)

  - Read/Write
  - bit map for mcast addr .132 .131 .130 .129 .107

    - bit[4]: 224.0.1.132
    - bit[3]: 224.0.1.131
    - bit[2]: 224.0.1.130
    - bit[1]: 224.0.1.129
    - bit[0]: 224.0.0.107

- /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/n/config

  (where n is slave port number)

  - Read Only
  - shows the raw values of the cpsw port ts register configurations

```
Examples:
```

```
1. Checking whether uni-cast enabled
   $ cat /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/uni_en
   $ 0
```

```
2. Enabling uni-cast
   $ echo 1 > /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/uni_en
```

```
3. Checking which multi-cast addr is enabled (when uni_en=0)
   $ cat /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/mcast_addr
   $ 0x1f
```

```
4. Disabling 224.0.1.131 and 224.0.0.107 but enabling the rest (when uni_en=0)
   $ echo 0x16 > /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/mcast_addr
```

```
5. Showing the current port time sync config
   $ cat /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/config
   000f06bb 001e88f7 81008100 01a088f7 00040000
```

```
   where the displayed hex values correspond to the port registers
   ts_ctl, ts_seq_ltype, ts_vlan_ltype, ts_ctl_ltype2 and ts_ctl2
```

Note 1: Although the above configurations are done through command line, they can also be done by using standard Linux open()/read()/write() file function calls.

Note 2: When uni-cast is enabled, ie. uni_en=1, mcast_addr configuration will not take effect since uni-cast will allow any uni-cast and multi-cast address.

CPTS Driver Internals Overview

1. Driver Initialization

On start up, the cpts driver

- initializes the input clock if it is an internal clock:

  - enable the input clock
  - get the clock frequency

- gets the frequency configuration of the input clock from the device tree bindings if it is an external clock
- selects/calculates (see Notes below for details) the multiplier (M), shift (S) and divisor (D) corresponding to the frequency for internal usage, ie. converting counter cycles to nsec by using the formula

nsec = ((cycles * M) >> S) / D

- gets the cpts_rftclk_sel value and program the CPTS RFTCLK_SEL register.
- configures the cpsw Px_TS_CTL, Px_TS_SEQ_LTYPE, Px_TS_VLAN_LTYPE, Px_TS_CTL_LTYPE2 and Px_TS_CTL2 registers (see section Configurations)
- registers itself to the Linux kernel ptp layer as a clock source (doing so makes sure the Linux kernel ptp layer and standard user space API's can be used)
- mark the currnet cpts counter value to the current system time
- schedule a periodic work to catch the cpts counter overflow events and updates the driver's internal time counter and cycle counter values accordingly.

Note 1: For a rftclk freq of 400MHz, the counter overflows at about every 10.73 secs. It is the responsibility of the software (ie. the driver) to keep track of the overflows and hence the correct time passed.

Note 2: The multiplier (M) shift (S) and divisor (D) depends on the rftclk frequency (F). Ideally, "good" values of M/S/D should be chosen so that when converting counter value when it reaches the rftclk frequency value (F) to timestamp time, i.e.

((F * M) >> S) / D

gives exactly 1000000000 nsec for accuracy and D should be 1 (if possible) to avoid long division for efficiency.

For example, if F = 614400000, to find M/S/D such that

1000000000 = 614400000 * M / (2^S * D)

simplify and rewrite both sides so that

2^4 * 5^4 = 2^11 * 3 * M / (2^S * D)

or

M / (2^S * D) = 5000 / (2^10 * 3)

hence

M = 5000, S = 10, D = 3

Note 3: cpts driver keeps a table of M/S/D for some common frequencies

Freq (Hz)	M	S	D
400000000	2560	10	1
425000000	5120	7	17
500000000	2048	10	1
600000000	5120	10	3
614400000	5000	10	3
625000000	4096	9	5
675000000	5120	7	27
700000000	5120	9	7
750000000	4096	10	3

Note 4: At start up, cpts driver selects or calculates the M/S/D for the rftclk frequency according to the following

a. if M/S/D is defined in devicetree bindings, use them; otherwise
b. if the rftclk frequency matches one of the frequencies in the table above, select the corresponding M/S/D; otherwise
c. if the rftclk frequency differs from one of the frequencies in the table above by less than 1 MHz, select the M/S/D that corresponds to the frequency with the minimum difference; otherwise
d. call clocks_calc_mult_shift( ) to calculate the M & S and set D = 1

Note 5: (**WARNING**) On Keystone 2 platforms, the default rftclk select is the internal SYSCLK2. On K2L, core pll is configured (based on the programmed efuse of max speed of 1 GHz and ref clk of 122880000 Hz) to 1000594244 Hz. As such, SYSCLK2 = 1000594244 / 2 = 500297122 Hz. With such a rftclk frequency, it is unlikely that some "good" M/S/D can be found so that 1000000000 = ((500297122 * M) >> S) / D. Hence based on the algorithm in Note 4, the M/S/D corresponding to 500000000 Hz will be used and unfortunately inaccuracy will be observed in timestamping. However, this issue is not observed on K2HK and K2E since the respective core pll is configured to exactly 1200000000 Hz and 1000000000 Hz, thus the cpts rftclk frequency is 600000000 and 500000000 Hz respectively and "good" M/S/D exist for these rftclk frequencies.

Note 6: Instead of an internal rftclk, cpts can be provided with an external rftclk. Also custom M/S/D can be configured in devicetree bindings.

2. Timestamping in Tx

In the tx direction during runtime, the driver

- marks the submitted packet to be CPTS timestamped if the the packet passes the PTP filter rules
- retrieves the timestamp on the transmitted ptp packet (packets submitted to a socket with proper socket configurations, see below) from CPTS's event FIFO
- converts the counter value to nsec (recall the internal time counter and the cycle counter kept internally by the driver)
- packs the retrieved timestamp with a clone of the transmitted packet in a buffer
- returns the buffer to the app which submits the packet for transmission through the socket's error queue

3. Timestamping in Rx

In the rx direction during runtime, the driver

- examines the received packet to see if it matches the PTP filter requirements
- if it does, then it retrieves the timestamp on the received ptp packet from the CPTS's event FIFO
- coverts the counter value to nsec (recall the internal time counter and the cycle counter kept internally by the driver)
- packs the retrieved timestamp with received packet in a buffer
- pass the packet buffer onwards

Using CPTS Timestamping

CPTS user applications use standard Linux APIs to send and receive PTP packets, and to adjust CPTS clock.

1. Send/receive L4 PTP messages (Annex D and E)

User application sends and receives L4 PTP messages by calling Linux standard socket API functions

```
Example (see Reference i):
```

```
    a. open UDP socket
    b. call ioctl(sock, SIOCHWTSTAMP, ...) to set the hw timestamping
       socket config
    c. bind to PTP event port
    d. set dst address to socket
    d. setsockopt to join multicast group (if using multicast)
    f. setsockopt to set socket option SO_TIMESTAMP
    g. sendto to send PTP packets
    h. recvmsg( ... MSG_ERRQUEUE ...) to receive timestamped packets
```

2. Send/receive L2 PTP messages (Annex F)

User application sends and receives PTP messages over Ethernet by opening Linux RAW sockets.

```
Example (see file raw.c in Reference iii):
```

```
    int fd
    fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    ...
```

In this case, PTP messages are encapsulated directly in Ethernet frames with EtherType 0x88f7.

3. Send/receive PTP messages in VLAN

When sending L2/L4 PTP messages over VLAN, **step b** in above example need to be applied to the actual interface instead of the VLAN interface.

```
Example (see Reference i):
```

```
Suppose a VLAN interface with vid=10 is added to the eth0 interface.
```

```
$ vconfig add eth0 10
$ ifconfig eth0.10 192.168.1.200
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:17:EA:F4:32:3A
          inet addr:132.168.138.88  Bcast:0.0.0.0  Mask:255.255.254.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:647798 errors:0 dropped:158648 overruns:0 frame:0
          TX packets:1678 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:58765374 (56.0 MiB)  TX bytes:84321 (82.3 KiB)
```

```
eth0.10   Link encap:Ethernet  HWaddr 00:17:EA:F4:32:3A
          inet addr:192.168.1.200  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::217:eaff:fef4:323a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:61 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:836 (836.0 B)  TX bytes:6270 (6.1 KiB)
```

```
To enable hw timestamping on the eth0.10 interface, the ioctl(sock, SIOCHWTSTAMP, ...)
function call needs to be on the actual interface eth0:
```

```
    int sock;
    struct ifreq hwtstamp;
    struct hwtstamp_config hwconfig;
```

```
    ...
```

```
    sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

```
    /* enable hw timestamping for interfaces eth0 or eth0.10 */
    strncpy(hwtstamp.ifr_name, "eth0", sizeof(hwtstamp.ifr_name));
    hwtstamp.ifr_data = (void *)&hwconfig;
    memset(&hwconfig, 0, sizeof(hwconfig));
    hwconfig.tx_type = HWTSTAMP_TX_ON
    hwconfig.rx_filter = HWTSTAMP_FILTER_PTP_V1_L4_SYNC
    ioctl(sock, SIOCSHWTSTAMP, &hwtstamp);
    ...
```

4. Clock Adjustments

User application needs to inform the CPTS driver of any time or reference clock frequency adjustments, for example, as a result of running PTP protocol.

- It's the application's responsibility to modify the (physical) rftclk frequency.
- However, the frequency change needs to be sent to the cpts driver by calling the standard Linux API clock_adjtime() with a flag ADJ_FREQUENCY. This is needed so that the CPTS driver can calculate the time correctly.
- As indicated above, CPTS driver keeps a pair of numbers, the multiplier and divisor, to represent the reference clock frequency. When the frequency change API is called and passed with the ppb change, the CPTS driver updates its internal multiplier as follows:

$$new\_mult = init\_mult + init\_mult * (ppb / 1000000000)$$

Note: the ppb change is always applied to the initial orginal frequency, NOT the current frequency.

```
Example (see Reference ii):
```

```
    struct timex tx;
    ...
    fd = open("/dev/ptp0", O_RDWR);
    clkid = get_clockid(fd);
    ...
    memset(&tx, 0, sizeof(tx));
    tx.modes = ADJ_FREQUENCY;
    tx.freq = ppb_to_scaled_ppm(adjfreq);
    if (clock_adjtime(clkid, &tx)) {
       perror("clock_adjtime");
    } else {
       puts("frequency adjustment okay");
    }
```

- To set time (due to shifting +/-), call the the standard Linux API clock_adjtime() with a flag ADJ_SETOFFSET

```
Example (see Reference ii):
```

```
    memset(&tx, 0, sizeof(tx));
    tx.modes = ADJ_SETOFFSET;
    tx.time.tv_sec = adjtime;
    tx.time.tv_usec = 0;
    if (clock_adjtime(clkid, &tx) < 0) {
       perror("clock_adjtime");
    } else {
       puts("time shift okay");
    }
```

- To get time, call the the standard Linux API clock_gettime()

```
Example (see Reference ii):
```

```
    if (clock_gettime(clkid, &ts)) {
       perror("clock_gettime");
    } else {
       printf("clock time: %ld.%09ld or %s",
              ts.tv_sec, ts.tv_nsec, ctime(&ts.tv_sec));
    }
```

- To set time, call the the standard Linux API clock_settime()

```
Example (see Reference ii):
```

```
    clock_gettime(CLOCK_REALTIME, &ts);
    if (clock_settime(clkid, &ts)) {
       perror("clock_settime");
    } else {
       puts("set time okay");
    }
```

Testing CPTS/PTP

To check the ptp clock adjustment with PTP protocol, a PTP slave (client) and a PTP master (server) applications are needed to run on separate devices (EVM or PC). Open source application package linuxptp (Reference iii) can be used as slave and as well as master. Another option for PTP master is the open source project ptpd (Reference iv).

- Slave Side Examples

The following command can be used to run a ptp-over-L4 client on the evm in slave mode

```
./ptp4l -E -4 -H -i eth0 -s -l 7 -m -q -p /dev/ptp0
```

For ptp-over-L2 client, use the command

```
./ptp4l -E -2 -H -i eth0 -s -l 7 -m -q -p /dev/ptp0
```

ptp4l runtime configuartions can be applied by saving desired configurations in a configuration file and start the ptp4l with an argument "-f <config_filename>" Note: Only ptp4l supports L2 ethernet, ptpd2 does not support L2. For example, put the following two lines

```
[global]
tx_timestamp_timeout  15
```

in a file named config, and start a ptp4l-over-L2 client with command

```
./ptp4l -E -2 -H -i eth0 -s -l 7 -m -q -p /dev/ptp0 -f config
```

the tx poll timeout interval will be set to 15 msec instead of the default 1 msec.

The adjusted time can be checked by cross compiling the testptp application from the linux kernel: Documentation/ptp/testptp.c. (e.g) ./testptp -g

- Master Side Examples

ptp4l can also be run in master mode. For example, the following command starts a ptp4l-over-L2 master on an EVM using **hardware timestamping**,

```
./ptp4l -E -2 -H -i eth0 -l 7 -m -q -p /dev/ptp0 -f config
```

On a Linux PC which does not supoort hardware timestamping, the following command starts a ptp4l-over-L2 master using **software timestamping**.

```
./ptp4l -E -2 -S -i eth0 -l 7 -m -q -p -f config
```

Who Is Timestamping What?

Notice that PA timestamping and CPTS timestamping are running simultaneously. This is desirable in some use cases because, for example, NTP timestamping is also needed in some systems and CPTS timestamping is only for PTP. However, CPTS has priority over PA to timestamp PTP messages. When CPTS timestamps a PTP message, PA will not timestamp it. See the section PA Timestamping for more details about PA timestamping.

If needed, PA timestamping can be completely disabled by adding force_no_hwtstamp to the device tree.

```
Example:
```

```
    pa: pa@2000000 {
            label = "keystone-pa";
            ...
            force_no_hwtstamp;
    };
```

CPTS timestamping can be completely disabled by removing the following line from the device tree

```
cpts_reg_ofs = <0xd00>;
```

Pulse-Per-Second (PPS)

The CPTS driver uses the timestamp compare (TS_COMP) output to support PPS.

The TS_COMP output is asserted for ts_comp_length[15:0] RCLK periods when the time_stamp value compares with the ts_comp_val[31:0] and the length value is non-zero. The TS_COMP rising edge occurs three RCLK periods after the values compare. A timestamp compare event is pushed into the event FIFO when TS_COMP is asserted. The polarity of the TS_COMP output is determined by the ts_polarity bit. The output is asserted low when the polarity bit is low.

1. CPTS Driver PPS Initialization

- The driver enables its pps support capability when it registers itself to the Linux PTP layer.
- Upon getting the pps support information from CPTS driver, the Linux PTP layer registers CPTS as a pps source with the Linux PPS layer. Doing so allows user applications to manage the PPS source by using Linux standard API.

2. CPTS Driver PPS Operation

- Upon CPTS pps being enabled by user application, the driver programs the TS_COMP_VAL for a pulse to be generated at the next (absolute) 1 second boundary. The TS_COMP_VAL to be programmed is calculated based on the reference clock frequency.
- Driver polls the CPTS event FIFO 5 times a second to retrieve the timestamp compare event of an asserted TS_COMP output signal.
- The driver reloads the TS_COMP_VAL register with a value equivalent to one second from the timestamp value of the retrieved event.
- The event is also reported to the Linux PTP layer which in turn reports to the PPS layer.

3. PPS User Application

- Enabling CPTS PPS by using standard Linux ioctl PTP_ENABLE_PPS

```
Example (Reference ii: Documentation/ptp/testptp.c):
```

```
    fd = open("/dev/ptp0", O_RDWR);
    ...
```

```
    if (ioctl(fd, PTP_ENABLE_PPS, 1))
         perror("PTP_ENABLE_PPS");
    else
         puts("pps for system time enable okay");
```

```
    if (ioctl(fd, PTP_ENABLE_PPS, 0))
         perror("PTP_ENABLE_PPS");
    else
         puts("pps for system time disable okay");
```

- Reading PPS last timstamp by using standard Linux ioctl PPS_FETCH

```
Example (Reference iii: linuxptp-1.2/phc2sys.c)
```

```
    ...
    struct pps_fdata pfd;
```

```
    pfd.timeout.sec = 10;
    pfd.timeout.nsec = 0;
    pfd.timeout.flags = ~PPS_TIME_INVALID;
    if (ioctl(fd, PPS_FETCH, &pfd)) {
       pr_err("failed to fetch PPS: %m");
       return 0;
    }
```

```
    ...
```

- Enabling PPS from sysfs
    - The Linux PTP layer provides a sysfs for enabling/disabling PPS.

```
    $ cat /sys/devices/soc.0/2090000.netcp/ptp/ptp0/pps_available
    1
    $ echo 1 > /sys/devices/soc.0/2090000.netcp/ptp/ptp0/pps_enable
```

- Sysfs Provided by Linux PPS Layer (see Reference v for more details)
    - The Linux PPS layer implements a new class in the sysfs for supporting PPS.

```
    $ ls /sys/class/pps/
    pps0/
    $
    $ ls /sys/class/pps/pps0/
    assert    clear  echo  mode  name  path  subsystem@  uevent
```

    - Inside each "assert" you can find the timestamp and a sequence number:

```
    $ cat /sys/class/pps/pps0/assert
    1170026870.983207967#8
```

```
    where before the "#" is the timestamp in seconds; after it is the sequence number.
```

4. Effects of Clock Adjustments on PPS

The user application calls the API functions clock_adjtime() or clock_settime() to inform the CPTS driver about any clock adjustment as a result of running the PTP protocol. The PPS may also need to be adjusted by the driver accordingly.

See **Clock Adjustments** in the CPTS User section for more details on clock adjustments.

- Shifting Time

    The user application informs CPTS driver of the shifts the clock by calling clock_adjtime() with a flag ADJ_SETOFFSET.

    Shifting time may result in shifting the 1 second boundary. As such the driver recalculates the TS_COMP_VAL for the next pulse in order to align the pulse with the 1 second boundary after the shift.

```
Example 1. Positive Shift
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

If no shifting happens, a pulse is asserted according to the following

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

Suppose a shift of +0.25 sec occurs at cntr=1458

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5               <- adjtime(ADJ_SETOFFSET, +0.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

Instead of going out at cntr=1508 (which was sec-13 but is now sec-13.25 after
the shift), a pulse will go out at cntr=1583 (or sec-14) after the
re-alignment at the 1-second boundary.

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75            (after +0.25 sec shift)
1483   13
1508   13.25            (realign orig pulse to cntr=1583)
1583   14      ^
1608   14.25
1683   15      ^
1708   15.25
.
.
.
```

Example 2. Negative Shift

Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).

If no shifting happens, a pulse is asserted according to the following

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

Suppose a shift of -3.25 sec occurs at cntr=1458

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5               <- adjtime(ADJ_SETOFFSET, -3.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

Instead of going out at cntr=1508 (which was sec-13 but is now sec-9.75
after the shift), a pulse will go out at cntr=1533 (or sec-10) after the
re-alignment at the 1-second boundary.

```
      (abs)
cntr   sec    pulse
----   ---    -----
```

```
1208   10      ^
1308   11      ^
1408   12      ^
1458   9.25            (after -3.25 sec shift)
1508   9.75            (realign orig pulse to cntr=1533)
1533   10      ^
1558   10.25
1608   10.75
1633   11      ^
1658   11.25
1708   11.75
.
.
.
```

Remark: If a second time shift is issued before the next re-aligned pulse is asserted after the first time shift, shifting of the next pulse can be accumulated.

```
Example 3. Accumulated Pulse Shift
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no shifting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose a shift of +0.25 sec occurs at cntr=1458
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5              <- adjtime(ADJ_SETOFFSET, +0.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-13.25 after
the shift), a pulse will go out at cntr=1583 (or sec-14) after the
re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75           (after +0.25 sec shift)
1483   13
1508   13.25           (realign orig pulse to cntr=1583)
1583   14      ^
1608   14.25
1683   15      ^
1708   15.25
.
.
.
```

```
Suppose another +0.25 sec time shift is issued at cntr=1533 before the
re-align pulse at cntr=1583 is asserted.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75
1483   13
1508   13.25
1533   13.5            <- adjtime(ADJ_SETOFFSET, +0.25 sec)
1583   14
1608   14.25
1683   15
1708   15.25
.
.
.
```

```
In this case the scheduled pulse at cntr=1583 is further shifted to cntr=1658.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75
1483   13
1508   13.25
1533   13.75             (after +0.25 sec shift)
1583   14.25
1608   14.5
1658   15      ^         (realign the cntr-1583-pulse to cntr=1658)
1683   15.25
1708   15.5
1758   16      ^
.
.
.
```

- Setting Time

  The user application may set the internal timecounter kept by the CPTS driver by calling clock_settime().

  Setting time may result in changing the 1-second boundary. As such the driver recalculates the TS_COMP_VAL for the next pulse in order to align the pulse with the 1 second boundary after the shift. The TS_COMP_VAL recalculation is similar to shifting time.

```
Example.
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no time setting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose at cntr=1458, time is set to 100.25 sec
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5              <- settime(100.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-100.75 after
the shift), a pulse will go out at cntr=1533 (or sec-101) after the
re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec      pulse
----   ---      -----
1208   10        ^
1308   11        ^
1408   12        ^
1458   100.25           (after setting time to 100.25 sec)
1508   100.75           (realign orig pulse to cntr=1533)
1533   101       ^
1608   101.75
1633   102       ^
1708   102.75
1733   103       ^
.
.
.
```

- Changing Reference Clock Frequency

  The user application informs the CPTS driver of the changes of the reference clock frequency by calling clock_adjtime() with a flag ADJ_FREQUENCY.

  In this case, the driver re-calculates the TS_COMP_VAL value for the next pulse, and the following pulses, based on the new frequency.

```
Example.
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no time setting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose at cntr=1458, reference clock freq is changed to 200Hz
```

```
*** Remark: The change to 200Hz is only for illustration.  The
            change should usually be parts-per-billion or ppb.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5               <- adjtime(ADJ_FREQUENCY, +100Hz)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-12.75 after
the freq change), a pulse will go out at cntr=1558 (or sec-13 in the new freq)
after the re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec      pulse
----   ---      -----
1208   10        ^
1308   11        ^
1408   12        ^
1458   12.5              (after freq changed to 200Hz)
1508   12.75             (realign orig pulse to cntr=1558)
1558   13        ^
1608   13.25
1658   13.5
1708   13.75
1758   14        ^
.
.
.
```

CPTS Hardware Timestamp Push

There are eight hardware time stamp inputs (HW1/8_TS_PUSH) that can cause hardware time stamp push events to be loaded into the event FIFO. The CPTS driver supports the reporting of such timestamps by using the PTP EXTTS feature of the Linux PTP infrastructure.

User applications can request such timestamps through ioctl() and read() function calls.

```
 Example (Reference ii: Documentation/ptp/testptp.c):
```

```
     struct ptp_extts_event event;
     struct ptp_extts_request extts_request;
```

```
     /* which pin to get timestamp from, index is 0 based */
     extts_request.index = 3;
     extts_request.flags = PTP_ENABLE_FEATURE;
```

```
     fd = open("/dev/ptp0", O_RDWR);
```

```
     /* enabling */
     ioctl(fd, PTP_EXTTS_REQUEST, &extts_request);
```

```
     /* reading timestamps */
     for (i=0; i < 10; i++) {
             read(fd, &event, sizeof(event));
             printf("event index %u at %lld.%09u\n", event.index,
                     event.t.sec, event.t.nsec);
     }
```

```
     /* disabling */
     extts_request.flags = 0;
```

```
    ioctl(fd, PTP_EXTTS_REQUEST, &extts_request);
```

Testing HW_TS_PUSH on Keystone2 (K2HK) EVM

Note: On K2HK EVM, only two HW_TS_PUSH pins are brought out. These are HW3_TS_PUSH and HW4_TS_PUSH. Refer to K2HK schematic for more details.

To use the TS_COMP_OUT signal to test HW_TS_PUSH:

1. Connect jumper pins CN17-5 (TSCOMPOUT_E) and CN17-3 (TSPUSHEVt0)
2. Connect pins CN3-114 (TSPUSHEVt0) and CN3-109 (TSPUSHEVt0_E). A ZX102-QSH 060-ST card is needed.
3. Modify testptp.c to "extts_request.index = 3", ie. reading timestamp from HW4_TS_PUSH pin
4. Compile testptp
5. Bootup K2HK Linux kernel
6. Under Linux prompt, issue "echo 1 > /sys/devices/soc.0/2090000.netcp/ptp/ptp0/pps_enable" to generate TS_COMP_OUT signals.
7. Under Linux prompt, issue "./testptp -e 10" to read the HW4_TS_PUSH timestamps.

CPTS References

i. Linux Documentation Timestamping Test (http://www.mjmwired.net/kernel/Documentation/networking/timestamping/timestamping.c)

ii. Linux Documentation PTP Test (http://www.mjmwired.net/kernel/Documentation/ptp/testptp.c)

iii. Open Source Project linuxptp (http://linuxptp.sourceforge.net)

iv. Open Source Project ptpd (http://ptpd.sourceforge.net)

v. Linux Documentation PPS (http://www.mjmwired.net/kernel/Documentation/pps/pps.txt)

vi. Linux pps-tools (https://github.com/ago/pps-tools)

Quality of Service

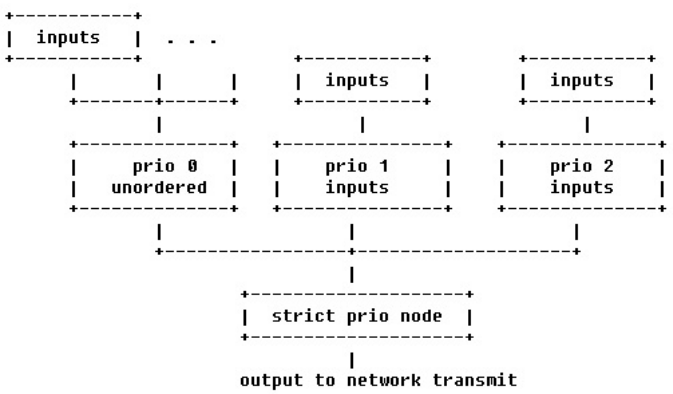
The linux hardware queue driver will download the Quality of Service Firmware to PDSP 1 of QMSS. PDSP 0 has accumulator firmware.

The firmware will be programmed by the linux keystone hardware queue QoS driver.

The configuration of the firmware is done with the help of device tree bindings. These bindings are documented in the kernel itself at **Documentation/devicetree/bindings/hwqueue/keystone-qos.txt**

QoS Tree Configuration

The QoS implementation allows for an abstracted tree of scheduler nodes represented in device tree form. An example is depicted below



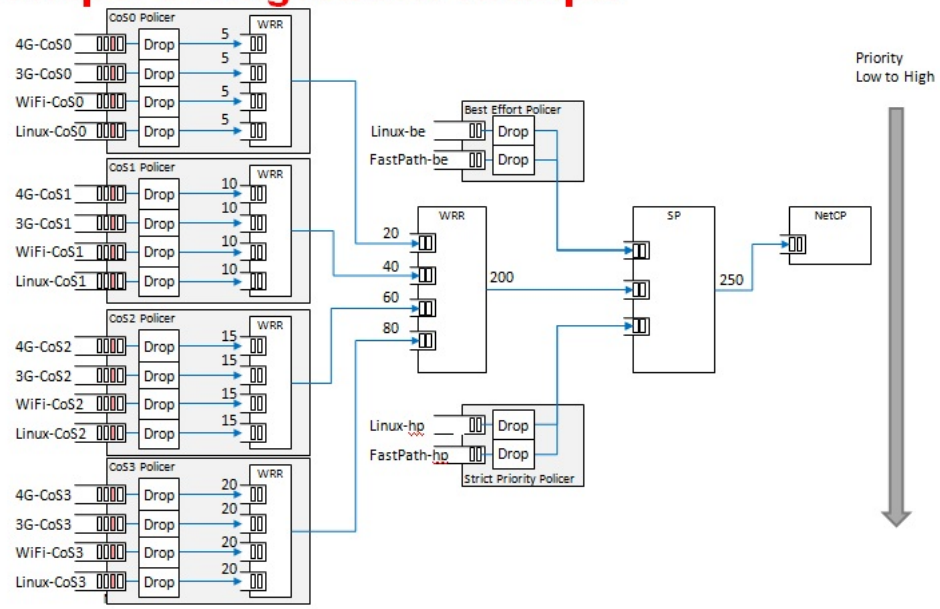
At each node, shaping and dropping parameters may be specified, within limits of the constraints outlined in this document. The following sections detail the device tree attributes applicable for this implementation.

The actual qos tree configuration can be found at linux/arch/arm/boot/dts/tci6638-evm.dts.

The device tree has attributes for configuring the QoS shaper. In the sections below we explain the various qos specific attributes which can be used to setup and configure a QoS shaper.

In the device tree we are setting up a shaper that is depicted below

Shaper configuration example



When egress shaper is enabled, all packets will be sent to the QoS firmware for shaping via a set of the queues starting from the QoS base queue which is 8000 by default. DSCP value in the IP header(outer IP incase of IPSec tunnels) or VLAN pbits (if VLAN interface) are used to determine the QoS queue to which the packet is sent. E.g., if the base queue is 8000, if the DSCP value is 46, the packet will be sent to queue number 8046. i.e., base queue number + DSCP value Incase of VLAN interfaces, if the pbit is 7, the packet will be sent to queue number 8071. i.e., base queue number + skip 64 queues used for DSCP + pbit value.

Shaper Configuration example, details

Flow ID	DSCP	VLAN p-bit	CoS	FastPath DSCP Queues 8000-8063	FastPath VLAN Queues 8064-8071	Linux CoS Queues 8072-8079	Flow Description
FastPath-hp	46	7		8046	8071	N/A	3gpp, high priority
4G-CoS3	34	4		8034	8068	N/A	4G Class3, platinum
3G-CoS3	36	4		8036			3G Class3, platinum
WiFi-CoS3	38	4		8038			WiFI Class3, platinum
4G-CoS2	26	3		8026	8067	N/A	4G Class2, gold
3G-CoS2	28	3		8028			3G Class2, gold
WiFi-CoS2	30	3		8030			WiFI Class2, gold
4G-CoS1	18	2		8018	8066	N/A	4G Class1, silver
3G-CoS1	20	2		8020,			3G Class, silver
WiFi-CoS1	22	2		8022			WiFI Class1, silver
4G-CoS0	10	1		8010	8065	N/A	4G Class0, bronze
3G-CoS0	12	1		8012			3G Class0, bronze
WiFi-CoS0	14	1		8014			WiFI Class0, bronze
FastPath-be	0+ all the rest	0+all the rest		8000+all the rest	8064+all the rest	N/A	3GPP, best effort
Linux-hp	Per DSCP to CoS mapping	Per p-bit to CoS mapping	5	N/A	N/A	8077	Linux high priority
Linux-CoS3			4	N/A	N/A	8076	Linux Class3, platinum
Linux-CoS2			3	N/A	N/A	8075	Linux Class2, gold
Linux-CoS1			2	N/A	N/A	8074	Linux Class1, silver
Linux-CoS0			1	N/A	N/A	8073	Linux Class0, bronze
Linux-be			0	N/A	N/A	8072+all the rest	Linux best effort

QoS Node Attributes

The following attributes are recognized within QoS configuration nodes:

- **"strict-priority" and "weighted-round-robin"**

e.g. strict-priority;

This attribute specifies the type of scheduling performed at a node. It is an error to specify both of these attributes in a particular node. The absence of both of these attributes defaults the node type to unordered(first come first serve).

- **"weight"**

e.g. weight = <80>;

This attribute specifies the weight attached to the child node of a weighted-round-robin node. It is an error to specify this attribute on a node whose parent is not a weighted-round-robin node.

- **"priority"**

e.g. priority = <1>;

This attribute specifies the priority attached to the child node of a strict-priority node. It is an error to specify this attribute on a node whose parent is not a strict-priority node. It is also an error for child nodes of a strict-priority node to have the same priority specified.

- **"byte-units" or "packet-units"**

e.g. byte-units;

The presence of this attribute indicates that the scheduler accounts for traffic in byte or packet units. If this attribute is not specified for a given node, the accounting mode is inherited from its parent node. If this attribute is not specified for the root node, the accounting mode defaults to byte units.

- **"output-rate"**

e.g. output-rate = <31250000 25000>;

The first element of this attribute specifies the output shaped rate in bytes/second or packets/second (depending on the accounting mode for the node). If this attribute is absent, it defaults to infinity (i.e., no shaping). The second element of this attribute specifies the maximum accumulated credits in bytes or packets (depending on the accounting mode for the node). If this attribute is absent, it defaults to infinity (i.e., accumulate as many credits as possible).

- **"overhead-bytes"**

e.g. overhead-bytes = <24>;

This attribute specifies a per-packet overhead (in bytes) applied in the byte accounting mode. This can be used to account for framing overhead on the wire. This attribute is inherited from parent nodes if absent. If not defined for the root node, a default value of 24 will be used. This attribute is passed through by inheritence (but ignored) on packet accounted nodes.

- **"output-queue"**

e.g. output-queue = <645>;

This specifies the QMSS queue on which output packets are pushed. This attribute must be defined only for the root node in the qos tree. Child nodes in the tree will ignore this attribute if specified.

- **"input-queues"**

e.g. input-queues = <8010 8065>;

This specifies a set of ingress queues that feed into a QoS node. This attribute must be defined only for leaf nodes in the QoS tree. Specifying input queues on non-leaf nodes is treated as an error. The absence of input queues on a leaf node is also treated as an error.

- **"stats-class"**

e.g. stats-class = "linux-best-effort";

The stats-class attribute ties one or more input stage nodes to a set of traffic statistics (forwarded/discarded bytes, etc.). The system has a limited set of statistics blocks (up to 48), and an attempt to exceed this count is an error. This attribute is legal only for leaf nodes, and a stats-class attribute on an intermediate node will be treated as an error.

- **"drop-policy"**

e.g. drop-policy = "no-drop"

The drop-policy attribute specifies a drop policy to apply to a QoS node (tail drop, random early drop, no drop, etc.) when the traffic pattern exceeds specifies parameters. The drop-policy parameters are configured separately within device tree (see "Traffic Police Policy Attributes section below). This attribute defaults to "no drop" for applicable input stage nodes. If a node in the QoS tree specifies a drop-policy, it is an error if any of its descendent nodes (children, children of children, ...) are of weighted-round-robin or strict-priority types.

Traffic Police Policy Attributes

The following attributes are recognized within traffic drop policy nodes:

- **"byte-units" or "packet-units"**

e.g. byte-units;

The presence of this attribute indicates that the dropr accounts for traffic in byte or packet units. If this attribute is not specified, it defaults to byte units. Policies that use random early drop must be of byte unit type.

- **"limit"**

e.g. limit = <10000>;

Instantaneous queue depth limit (in bytes or packets) at which tail drop takes effect. This may be specified in combination with random early drop, which operates on average queue depth (instead of instantaneous). The absence of this attribute, or a zero value for this attribute disables tail drop behavior.

- **"random-early-drop"**

e.g. random-early-drop = <32768 65536 2 2000>;

The random-early-drop attribute specifies the following four parameters in order:

low threshold: No packets are dropped when the average queue depth is below this threshold (in bytes). This parameter must be specified.

high threshold: All packets are dropped when the average queue depth above this threshold (in bytes). This parameter is optional, and defaults to twice the low threshold.

max drop probability: the maximum drop probability

half-life: Specified in milli seconds. This is used to calculate the average queue depth. This parameter is optional and defaults to 2000.

Sysfs support

The keystone hardware queue driver has sysfs support for statistics, drop policies and the tree configuration.

```
root@keystone-evm:/sys/devices/soc.0/hwqueue.8/qos-inputs-1# ls
drop-policies  qos-tree       statistics
root@keystone-evm:/sys/devices/soc.0/hwqueue.8/qos-inputs-1#
```

The above shows the location in the kernel where sysfs entries for the keystone hardware queue can be found. There are sysfs entries for the qos trees (qos-inuputs1, qos-tree-inputs2). Within the qos directory there are separate directories for statistics, drop-policies and the qos-tree itself. Each node in the tree is a separate directory entry, starting with the root (tip) entry.

Statistics are displayed for each statistics class in the device tree. Four statistics are represented for each stats class.

- bytes forwarded
- bytes discarded
- packets forwarded
- packets discarded

An example is depicted below

```
cat /sys/devices/soc.0/hwqueue.8/qos-inputs-1/statistics/linux-be/packets_forwarded
```

Drop policy configuration is also displayed for each drop policy. In the case of a drop policy, the parameters can also be changed. This is depicted below. Please note the the parameters that can be modified for tail drop are a subset of the parameters that can be modified for random early drop.

The qos tree is reached via the **qos_tree** directory and its sub-directories. Each sub-directory entry may contain:

- directory entries to reach the subtrees feeding this node
- the input queues to this node (valid for leaf nodes only)
- the output queue from this node
- the output rate for the node. The current value can be shown by: "cat output_rate". The value can be modified by: *echo "<val>" > output_rate*
- the overhead bytes parameter for the node. The current value can be shown by: "cat overhead_bytes". The value can be modified by: *echo "<val>" > overhead_bytes*
- burst size . The current value can be shown by: "cat burst_size". The value can be modified by: *echo "<val>" > burst_size*
- drop_policy . This is the name of the drop policy to be used.
- stats_class associated with node. This is the name of stats class to be used
- the priority of the node (for strict priority nodes only). The current value can be shown by: "cat priority". The value can be modified by: *echo "<val>" > priority*
- weight : for wrr nodes. The current value can be shown by: "cat weight". The value can be modified by: *echo "<val>" > weight*

Debug Filesystem support

Debug Filesystem(debugfs) support is also being provided for QoS support. To make use of debugfs support a user might have to mount a debugfs filesystem. This can be done by issuing the command.

```
mount -t debugfs debugfs /debug
```

The appropriate path and contents are shown below

```
root@keystone-evm:/debug/qos-1# ls
config_profiles  out_profiles     queue_configs    sched_ports
```

With the debugfs support we will be able to see the actual configuration of

- QoS scheduler ports
- Drop scheduler queue configs
- Drop scheduler output profiles
- Drop scheduler config profiles

The QoS scheduler port configuration can be seen by issuing the command **cat /debug/qos/sched_ports**. This is shown below

```
root@keystone-evm:/debug/qos-1# cat sched_ports
port 14
unit flags 15 group # 1 out q 8171 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 15
unit flags 15 group # 1 out q 8170 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 16
unit flags 15 group # 1 out q 8169 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 17
unit flags 15 group # 1 out q 8168 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 18
unit flags 15 group # 1 out q 8173 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 2457600
queue 1 cong thresh 0 wrr credit 4915200
queue 2 cong thresh 0 wrr credit 7372800
queue 3 cong thresh 0 wrr credit 9830400
```

```
port 19
unit flags 15 group # 1 out q 645 overhead bytes 24 throttle thresh 3125 cir credit 6400000 cir max
51200000
total q's 3 sp q's 3 wrr q's 0
queue 0 cong thresh 0 wrr credit 0
queue 1 cong thresh 0 wrr credit 0
queue 2 cong thresh 0 wrr credit 0
```

```
root@keystone-evm:/debug/qos-1#
```

cat command can be used in a similar way for displaying the Drop scheduler queue configs, output profiles and config profiles

Configuring QoS on an 1-GigE interface

To configure QoS on an interface, several definitions must be added to the device tree:

- Drop policies and a QoS tree must be defined. The outer-most QoS block must specify an output queue number; this may be the 1-GigE NETCP's PA PDSP 5 (645) or CPSW (648), one of the 10-GigE CPSW's queues (8752, 8753), or other queue as appropriate.

```
Example (k2hk-evm.dts):
```

```
droppolicies: default-drop-policies {
        no-drop {
                default;
                packet-units;
                limit = <0>;
        };
        ...
        all-drop {
                byte-units;
                limit = <0>;
        };
};
```

```
Example (k2hk-evm.dts):
```

```
qostree: qos-tree {
        strict-priority;                /* or weighted-round-robin */
        byte-units;                     /* packet-units or byte-units */
        output-rate = <31250000 25000>;
        overhead-bytes = <24>;          /* valid only if units are bytes */
        output-queue = <645>;           /* allowed only on root node */
```

```
        high-priority {
                ...
        }
        ...
        best-effort {
                ...
        };
};
```

```
qostree2: qos-tree-2 {
        strict-priority;                /* or weighted-round-robin */
        byte-units;                     /* packet-units or byte-units */
        output-rate = <31250000 25000>;
        overhead-bytes = <24>;          /* valid only if units are bytes */
        output-queue = <648>;           /* allowed only on root node */
```

```
        high-priority {
                ...
        }
        ...
        best-effort {
                ...
        };
};
```

- QoS inputs must be defined to the hwqueue subsystem. The QoS inputs block defines which group of hwqueues will be used, and links to the set of drop policies and QoS tree to be used.

```
Example (k2hk-evm.dts):
```

```
hwqueue0: hwqueue@2a40000 {
        ...
        queues {
                ...
                qos-inputs-1 {
                        values                  = <8000 192>;
                        pdsp-id                 = <3>;
                        ...
                        drop-policies           = <&droppolicies>;
                        qos-tree                = <&qostree>;
                        reserved;
                };
                qos-inputs-2 {
                        values                  = <6400 192>;
                        pdsp-id                 = <7>;
                        ...
                        drop-policies           = <&droppolicies>;
                        qos-tree                = <&qostree2>;
                        reserved;
                };
        };
}; /* hwqueue0 */
```

- A PDSP must be defined, and loaded with the QoS firmware.

```
Example (k2hk-evm.dts):
```

```
hwqueue0: hwqueue@2a40000 {
       ...
       pdsps {
               ...
               pdsp3@0x2a13000 {
                       firmware = "keystone/qmss_pdsp_qos_k2_le_2_0_1_5.fw";
                       ...
                       id = <3>;
               };
               pdsp7@0x2a17000 {
                       firmware = "keystone/qmss_pdsp_qos_k2_le_2_0_1_5.fw";
                       ...
                       id = <7>;
               };
       };
}; /* hwqueue0 */
```

- A Packet DMA channel must be defined and associated with each of the QoS input queues

```
Example (k2hk-evm.dts):
```

```
padma: pktdma@2004000 {
        ...
        channels {
                ...
                qos0 {
                        transmit;
                        label           = "qos0";
                        pool            = "pool-net";
                        submit-queue    = <8072>;
                        ...
                };
                ...
                qos5 {
                        transmit;
                        label           = "qos5";
                        pool            = "pool-net";
                        submit-queue    = <8077>;
                        complete-queue  = <8707>;
```

```
                ...
        };
        qos6 {
                transmit;
                label           = "qos6";
                pool            = "pool-net";
                submit-queue    = <6472>;
                ...
        };
        ...
        qos11 {
                transmit;
                label           = "qos11";
                pool            = "pool-net";
                submit-queue    = <6477>;
                complete-queue  = <8708>;
                ...
        };
    };
};
```

- A NETCP QoS block must be defined. For each interface and "interface-x" block is defined, which contains definitions for each of the QoS input DMA channels to be associated with that interface.

```
Example (k2hk-evm.dts):
```

```
netcp: netcp@2090000 {
    ...
    qos: qos@0 {
        label = "keystone-qos";
        multi-interface;
```

```
        interface-0 {
                chan-0 {
                        tx-channel = "qos0";
                        tx_queue_depth = <64>;
                };
                ...
                chan-5 {
                        tx-channel = "qos5";
                        tx_queue_depth = <64>;
                };
        };
        interface-1 {
                chan-0 {
                        tx-channel = "qos6";
                        tx_queue_depth = <64>;
                };
                ...
                chan-5 {
                        tx-channel = "qos11";
                        tx_queue_depth = <64>;
                };
        };
    };
};
```

- By default, Linux network traffic will be queued to the interface "chan-0" subqueue. To classify and route packets from Linux to specific QoS queues, the Linux traffic control utility "tc" must be used. First a class-full root queuing discipline must be established for the interface, and then filters may be used to classify packets. These filters can use the "skbedit queue_mapping" action to set the subqueue number for the packet. Here is an example:

```
# Clear any existing configuration
tc qdisc del dev eth0 root
```

```
# Add DSMARK as the root qdisc
tc qdisc add dev eth0 root handle 1 dsmark indices 8 default_index 0
```

```
# Create filters to classify packets and route to queues
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5002 0xffff \
        action skbedit queue_mapping 1
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5003 0xffff \
        action skbedit queue_mapping 2
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5004 0xffff \
        action skbedit queue_mapping 3
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5005 0xffff \
        action skbedit queue_mapping 4
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5006 0xffff \
        action skbedit queue_mapping 5
```

Please refer to the Linux Advanced Routing & Traffic Control how-tos and related manpages available on the Internet for more information on "tc".

Disabling QoS on an 1-GigE interface

The released "k2hk-evm.dts" file contains definitions for two QoS trees and associates them with the first two ports on the 1-GigE interface. These default trees are configured so that traffic queued to interface subqueue 0 will bypass the QoS tree. Only traffic specifically directed to subqueues 1-6 will be processed through the hardware QoS subsystem. This may be sufficient for your needs. However, you may prefer to remove the QoS configuration entirely from the device tree.

To disable QoS on the two 1-GigE interfaces

- delete all the qos related blocks or entries shown in the examples in section Configuring QoS on an 1-GigE interface, namely
    - droppolicies: default-drop-policies {...}
    - qostree: qos-tree {...}
    - qostree2: qos-tree-2 {...}
    - qos-inputs-1 {...}
    - qos-inputs-2 {...}

- pdsp3@0x2a13000 {...}
- pdsp7@0x2a17000 {...}
- qos0 {...} to qos11 {...}
- qos: qos@0 {...}

- modify tx_data_queue_depth in the pa: pa@2000000 {...} block in k2hk-evm.dts to 256.

Configuring QoS on a 10-GigE interface

The following snippets together shows how to remove the QoS tree associated with the second port of the 1-GigE interface and associate it with the first port on the 10-GigE interface. In these snippets, we only depict and highlight the modifications made to the above 1-GigE examples. Contents not shown in the definitions should just be copy and paste from the file k2hk-evm.dts.

Note: this is only for demonstration purpose and is not part of the release.

- Modify the output-queue number of qostree2 to that of the transmit queue of the 10-GigE's first port.

```
qostree2: qos-tree-2 {
        strict-priority;                /* or weighted-round-robin */
        byte-units;                     /* packet-units or byte-units */
        output-rate = <31250000 25000>;
        overhead-bytes = <24>;          /* valid only if units are bytes */
        output-queue = <8752>;           /* allowed only on root node */
```

```
        high-priority {
                ...
        }
        ...
        best-effort {
                ...
        };
};
```

- Remove qos6 to qos11 from padma's channels block.

```
padma: pktdma@2004000 {
         ...
         channels {
                ...
                qos0 {
                        transmit;
                        label           = "qos0";
                        pool            = "pool-net";
                        submit-queue    = <8072>;
                        ...
                };
                ...
                qos5 {
                        transmit;
                        label           = "qos5";
                        pool            = "pool-net";
                        submit-queue    = <8077>;
                        complete-queue  = <8707>;
                        ...
                };
                /* qos6 - qos11 removed */
        };
};
```

- Define qos6 to qos11 in xgedma's channels block. This is a cut-and-paste of those defined in the padma block but with the pool name pool-**net** modified to pool-**xge**.

```
xgedma: pktdma@2fa1000 {
         ...
         channels {
                ...
                qos6 {
                        transmit;
                        label           = "qos6";
                        pool            = "pool-xge";
                        submit-queue    = <6472>;
                        ...
                };
                ...
                qos11 {
                        transmit;
                        label           = "qos11";
                        pool            = "pool-xge";
                        submit-queue    = <6477>;
                        complete-queue  = <8708>;
                        ...
                };
        };
};
```

- Remove interface-1 from netcp's qos block.

```
netcp: netcp@2090000 {
        ...
        qos: qos@0 {
                label = "keystone-qos";
                multi-interface;
```

```
                interface-0 {
                        chan-0 {
                                tx-channel = "qos0";
                                tx_queue_depth = <64>;
                        };
                        ...
                        chan-5 {
                                tx-channel = "qos5";
                                tx_queue_depth = <64>;
                        };
                };
```

```
                /* interface-1 removed */
        };
};
```

- Define a qos block qos**x**: qos@0 {...} in 10-GigE's netcp block. The chan-N content of this block is the same as those defined in the interface-1 of netcp's qos.

```
netcpx: netcp@2f00000 {
        ...
        qosx: qos@0 {
                label = "keystone-qos";
                multi-interface;
```

```
                interface-0 {
                        chan-0 {
                                tx-channel = "qos6";
                                tx_queue_depth = <64>;
                        };
                        ...
                        chan-5 {
                                tx-channel = "qos11";
                                tx_queue_depth = <64>;
                        };
                };
        };
};
```

Crypto Driver

Keystone SoC includes a hardware cryptographic accelerator engine also known as CP_ACE (Communication Processor Adaptive Cryptographic engine). The CP_ACE subsystem is designed to provide packet security as part of IPSec, SRTP and 3GPP industry standards. Keystone Linux kernel implements a crypto driver which offloads crypto algorithm processing to CP_ACE. Crypto driver registers algorithm implementations in the kernel's crypto algorithm management framework. Since the primary use case for this driver is IPSec ESP offload, it currently registers only AEAD algorithms. Following algorithms are supported by the driver:

1. authenc(hmac(sha1),cbc(aes))
2. authenc(hmac(sha1),cbc(des3-ede))
3. authenc(xcbc(aes),cbc(aes))
4. authenc(xcbc(aes),cbc(des3-ede))

The driver code can be found in the files drivers/crypto/keystone-sa.[ch]

USB Driver

The USB 3.0 xHC controller supports the hardware part of the xHCI standard and is implemented with the Synopsys DesignWare core. Driver support of the hardware is provided by the generic usb driver code found in directories and files drivers/usb/, drivers/usb/host/xhci*.c and drivers/usb/dwc3/. Platform specific XHCI glue code is implemented in drivers/usb/dwc3/dwc3-keystone.c and drivers/usb/phy/phy-keystone.c. Currently only usb host mode is supported.

K2E has 2 USB controllers, USB0 and USB1. This USB driver supports both USB controllers simultaneously in host mode. However, the USB connector for USB1 on K2E EVM is an USB 3.0 micro AB connector, adapter is needed for using it in host mode. Because of that, the devicetree bindings for USB1 is disabled by default. To enable the USB1 devicetree bindings, do one of the following:

- Enabling K2E USB1 at Compile Time

  Update the **status** property of usb1_phy and usb1 bindings from "disabled" to "ok" in arch/arm/boot/dts/k2e.dtsi:

  usb1_phy: usb_phy@2620750 {

  status = "**ok**";

  };

  usb1: usb@25000000 {

  status = "**ok**";

  };

- Enabling K2E USB1 at Bootup Time

  Alternately, the above updates can be done right before kernel bootup through u-boot scripting.

  setenv run_fdt_usb1 'fdt addr ${addr_fdt}; fdt set /soc/usb_phy@2620750 status "ok"; fdt set /soc/usb@25000000 status "ok" '
  setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon **run_fdt_usb1** run_kern'

10Gig Ethernet Driver

The TI Keystone-2 platforms include a 10 Gig Ethernet Subsystem. The subsystem combines a 3-port ethernet switch sub-module and a packet DMA capable of 10Gbps and 1Gbps per ethernet port. The ethernet switch sub-module contains an EMAC module, SGMII modules, 10GBase-R module, SERDES module, MACSec module and MDIO module.

The 10Gig Ethernet Driver makes use of the multi-instance support feature of the NETCP driver which serves as a hook to the Linux Network Stack. The 10Gig Ethernet Driver provides its API to the NETCP driver by registering itself to an instance of the NETCP driver during kernel initialization.

Refer to the Network Driver section for more details about the NETCP driver model. But notice that the packet accelerator in that section is not applicable to the 10Gig Ethernet subsystem.

The 10Gig Ethernet Driver code can be found in files keystone_xgess.c, keystone_xgepcsr.c and keystone_xgemdio.c in directory drivers/net/ethernet/ti/.

For releases **before MCSDK 3.0.2** release, the 10Gig ethernet driver **was not tested due to lack of hardware**. In those releases the driver is not included in the kernel default configuration. To enable the 10Gig Ethernet Driver, add the following two lines in file arch/arm/configs/keystone2_defconfig or arch/arm/configs/keystone2_fullrt_defconfig, and do a distclean build.

```
CONFIG_TI_KEYSTONE_XGE=y
CONFIG_TI_KEYSTONE_XGE_MDIO=y
```

Starting from MCSDK 3.0.2 release, the following lines are added to arch/arm/configs/keystone2_defconfig and arch/arm/configs/keystone2_fullrt_defconfig to enable 10Gig ethernet driver by default.

```
CONFIG_TI_KEYSTONE_XGE=y
CONFIG_MDIO_BITBANG=y
CONFIG_MDIO_GPIO=y
CONFIG_GPIO_PCA953X=y
```

Enabling 10Gig Ethernet Driver Device Tree Bindings

Starting from MCSDK 3.1.0 release, the 10Gig related device tree bindings are disabled by default. Depending on the link interface type, the 10Gig device tree bindings can be enabled for K2E and K2HK platforms at compile time or bootup time as follows.

MAC-to-PHY Link Interface

For example, when a TI K2HK or K2E EVM is connected to a **RevA RTM-BOC with 10Gig PHY**.

- Enabling 10Gig at Compile Time

  Update the **status** property of mdiox and netcpx bindings from "disabled" to "ok" in arch/arm/boot/dts/k2e-net.dtsi or arch/arm/boot/dts/k2hk-net.dtsi:

  mdiox0: mdiox {

  status = "**ok**";

  };

  netcpx: netcp@2f00000 {

  status = "**ok**";

  };

  Also update the **status** property of pca@20 binding from "disabled" to "ok" in arch/arm/boot/dts/keystone.dtsi:

  pca@20 {

  status = "**ok**";

  };

- Enabling 10Gig at Bootup Time

  Alternately, the above updates can be done right before kernel bootup through u-boot scripting.

  setenv run_fdt 'fdt addr ${addr_fdt}; fdt set /soc/i2c2/pca@20 status "ok"; fdt set /soc/netcp@2f00000 status "ok"; fdt set /soc/mdiox/ status "ok" '
  setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon **run_fdt** run_kern'

MAC-to-MAC Link Interface

For example, when a TI K2HK or K2E EVM is connected to a **Rev.B/C RTM-BOC with Dual Retimer**.

- Enabling 10Gig at Compile Time

  Update the **status** property of netcpx binding from "disabled" to "ok" and set each slave's **link-interface** type to 11 in arch/arm/boot/dts/k2e-net.dtsi or arch/arm/boot/dts/k2hk-net.dtsi:

  netcpx: netcp@2f00000 {

  status = "**ok**";
  slaves {

  slave0 {

  link-interface = <**11**>;
  phy-handle = <&phyx0>;

  };
  slave1 {

```
                        link-interface = <11>;
                        phy-handle = <&phyx1>;

                    };

                };

            };
```

Note: The phy-handle in each slave node can optionally be removed. phy-handle will not take effect when the link-interface is MAC-to-MAC.

- Enabling 10Gig at Bootup Time

  Alternately, the above updates can be done right before kernel bootup through u-boot scripting.

```
setenv run_fdt_1 'fdt addr ${addr_fdt}; fdt set /soc/netcp@2f00000 status "ok"'
setenv run_fdt_2 'fdt set /soc/netcp@2f00000/cpswx@2f00000/slaves/slave0 link-interface <11>'
setenv run_fdt_3 'fdt set /soc/netcp@2f00000/cpswx@2f00000/slaves/slave1 link-interface <11>'
setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon run_fdt_1 run_fdt_2 run_fdt_3
run_kern'
```

The driver has been verified on Keystone K2HK and K2E EVMs connected to RevA RTM-BOC with 10Gig PHY or RevB RTM-BOC with Dual Retimer.

Disabling Support of 10-GigE

To disable the support of 10Gig ethernet,

- Remove the 10-GigE configurations from the defconfig file arch/arm/configs/keystone2_defconfig or arch/arm/configs/keystone2_fullrt_defconfig
- Remove the following bindings from the file arch/arm/boot/dts/k2hk-evm.dts
  - xge{...} and pool-xge {...} in hwqueue0
  - xgedma: pktdma@2fa1000 {...}
  - mdiox0: mdiox {...}
  - netcpx: netcp@2f00000 {...}

Enabling 10G-KR Firmware

Starting from MCSDK 3.1.4, 10G-KR firmware can be enabled for handling of auto-negotiation and link training.

- Enabling firmware through DTS:

  Update the status property of the firmware and lane bindings from "disabled" to "ok" in arch/arm/boot/dts/k2e-net.dtsi or arch/arm/boot/dts/k2hk-net.dtsi:

```
firmware {

    status = "ok";
    lane0 {

        status = "ok";

    };
    lane1 {

        status = "ok";

    };
```

  Additionally, configure the **link-interface** type to MAC-to-MAC using the above steps.

Refer to the keystone-serdes.txt under Documentation/ for more information on device tree options.

10G-KR Firmware Usage Notes

- Due to constraints there are several usage notes concerning the firmware:

1. When autonegotiation occurs there is a reset asserted on the lane that affects the MAC layer and switch.
   1. During a simultaneous boot of two devices they will sync and autonegotiate before the aforementioned layers are configured. There is no issue in this scenario.
   2. If a single device is reset this will cause autonegotiation to occur again. This will reset the lane of the device that stayed persistently on. When this happens, re-program the MAC_CONTROL register for that lane, otherwise, an interface toggle using 'ifconfig' is sufficient to reconfigure the interface back to a working state.
2. When switching between a non-FW configuration and a FW configuration a POR is required.
3. Due to errata KeyStoneII.BTS_errata_advisory.29:10GbE PCS Causes Data Corruption, occasionally on link negotiation there may be high levels of packet loss.
   1. The symptoms of this are high packet loss, CRC and alignment errors, and 0xff block errors in a small time period.
   2. When this case is detected, you must reset the PCSR following the below procedure, where iBase is the base of the 10GE SerDes, 0x0231e000.

```
void fPcsrLaneResetToggle(unsigned int iBase, int iLane)

{

        uint32_t *addr = (uint32_t *)(iBase + 0x01ff4);        //This function assumes no one else is playing with this register.
        uint32_t iOriginal,iTmp;

        iOriginal = *addr;                      //Read Current value
        iTmp = iOriginal;                       //Make a copy

        if (((iOriginal>>iLane)&1)==0) return;  //No signal, the PCSR is held in reset!

        iTmp |= 0x00f0;                       //Set all continue bits
```

```
        *addr = iTmp;                       //Write out the register to set all lane signal continue bits
        iTmp  = *addr;                      //Read the register, Allows the cont_sd to propagate
        iTmp &= ~(1<<(iLane + 4));          //Clear the lane Cont_SD bit
        iTmp |= 1<<16;                      //Set the wafter_sd bit, since all SD cont_bits are set already, a glitch will not be seen.
        *addr = iTmp;                       //Write out the register to clear lane signal detect
        iTmp  = *addr;                      //Read the register, Allows the cont_sd to propagate
        fSerdesSbCycleDelay(10000);         //>100nS wait.
        *addr = iOriginal;                  //Write the original value back to the register.
}
```

1. 1. The firmware can be restarted by writing to CPU_CTRL register, POR_EN bit 29. Set this bit high, then set it low with at least 10ms in between.

- For helping with debugging there is a tool available called serdes_print. As of right now this is not on an external portal, so it will have to be requested.

    The options available in this utility are:
    a -print all.
    b -dump the tbus
    p -print 10Gb phyb memory.
    m -print serdes registers.
    v -print adaptation values.
    x -print 10Gb switch registers.
    t -print xdata history.
    z -do a program counter trace.
    k -print some KR status registers.
    T -print transmitter parameters.
    R -print receiver parameters.

PCIe Driver

The TI Keystone platforms contain a PCIe module which supports a single bi-directional link interface with a maximum of two lanes width (x2). The module supports Root Complex and End Point operation modes.

The PCIe driver implemented supports only the Root Complex (RC) operation mode on K2 platforms (K2HK, K2E). An irq-chip is implemented to support both legacy and MSI irqs. The driver's main implementation can be found in drivers/pci/host/pcie-keystone.c and platform specific code can be found at drivers/pci/host/k2-platform.c

PCIe has been verified on K2E and K2HK EVMs. K2E supports two PCI ports. Port 0 is on Domain 0 and Port 1 is on Domain 1. On K2E EVM, a Marvel SATA controller, 0x9182 is connected to port 1 that supports interfacing with Hard disk drives (HDD). Following h/w setup is used to test SATA HDD interface with K2E. Western Digital 1.0 TB SATA / 64MB Cache hard disk drive, WD10EZEX is used for the test over PCI port 1.

Connect HDD to an external power supply. Connect the HDD SATA port to K2E EVM SATA port using a 6Gbps data cable and power on the HDD. Power On K2E EVM. The K2E rev 1.0.2.0 requires a hardware modification to get the SATA detection on the PCI bus. Please check with EVM hardware vendor for the details.

At boot up, the PCI enumeration should display the SATA EP detection.

```
[341130.650587] keystone-pcie: Starting PCI scan, nr 0...
[341130.650728] PCI host bridge to bus 0001:00
[341130.650737] pci_bus 0001:00: root bus resource [mem 0x60000000-0x6fffffff]
[341130.650744] pci_bus 0001:00: root bus resource [io  0x4000-0x7fff]
[341130.650751] pci_bus 0001:00: No busn resource found for root bus, will use [bus 00-ff]
[341130.651003] PCI: bus0: Fast back to back transfers disabled
[341130.651011] pci 0001:00:00.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[341130.651519] PCI: bus1: Fast back to back transfers disabled
[341130.651548] keystone-pcie: Ending PCI scan...
[341130.651556] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651560] keystone-pcie: keystone_pcie_map_irq: legacy_irq 572
[341130.651570] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 2
[341130.651574] keystone-pcie: keystone_pcie_map_irq: legacy_irq 573
[341130.651584] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651588] keystone-pcie: keystone_pcie_map_irq: legacy_irq 572
[341130.651597] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651601] keystone-pcie: keystone_pcie_map_irq: legacy_irq 608
[341130.651609] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651613] keystone-pcie: keystone_pcie_map_irq: legacy_irq 608
[341130.651631] pci 0001:00:00.0: BAR 8: assigned [mem 0x60000000-0x601fffff]
[341130.651639] pci 0001:00:00.0: BAR 9: assigned [mem 0x60200000-0x603fffff pref]
[341130.651647] pci 0001:00:00.0: BAR 7: assigned [io  0x4000-0x4fff]
[341130.651657] pci 0001:01:00.0: BAR 6: assigned [mem 0x60200000-0x6020ffff pref]
[341130.651665] pci 0001:01:00.0: BAR 5: assigned [mem 0x60000000-0x600001ff]
[341130.651676] pci 0001:01:00.0: BAR 4: assigned [io  0x4000-0x400f]
[341130.651686] pci 0001:01:00.0: BAR 0: assigned [io  0x4010-0x4017]
[341130.651695] pci 0001:01:00.0: BAR 2: assigned [io  0x4018-0x401f]
[341130.651705] pci 0001:01:00.0: BAR 1: assigned [io  0x4020-0x4023]
[341130.651715] pci 0001:01:00.0: BAR 3: assigned [io  0x4024-0x4027]
[341130.651725] pci 0001:00:00.0: PCI bridge to [bus 01]
[341130.651732] pci 0001:00:00.0:   bridge window [io  0x4000-0x4fff]
[341130.651740] pci 0001:00:00.0:   bridge window [mem 0x60000000-0x601fffff]
[341130.651748] pci 0001:00:00.0:   bridge window [mem 0x60200000-0x603fffff pref]
[341130.651757] PCI: enabling device 0001:00:00.0 (0140 -> 0143)
[341130.651764] keystone-pcie: keystone_pcie_rc_init - end
```

And then look for AHCI and SCSI logs as

```
[   15.651707] ahci 0001:01:00.0: AHCI 0001.0000 32 slots 2 ports 6 Gbps 0x3 impl SATA mode
[   15.658443] ahci 0001:01:00.0: flags: 64bit ncq sntf led only pmp fbs pio slum part sxs
[   15.666086] scsi0 : ahci
[   15.668410] scsi1 : ahci
[   15.670689] ata1: SATA max UDMA/133 abar m512@0x60000000 port 0x60000100 irq 612
[   15.676869] ata2: SATA max UDMA/133 abar m512@0x60000000 port 0x60000180 irq 612
```

Very first time with no partition and file format on HDD, following logs will be displayed

```
[   15.967048] ata1: SATA link down (SStatus 0 SControl 300)
[   16.142035] ata2: SATA link up 6.0 Gbps (SStatus 133 SControl 300)
[   16.147646] ata2.00: ATA-9: WDC WD10EZEX-08M2NA0, 01.01A01, max UDMA/100
[   16.153230] ata2.00: 1953525168 sectors, multi 0: LBA48 NCQ (depth 31/32), AA
[   16.159659] ata2.00: configured for UDMA/100
[   16.163344] scsi 1:0:0:0: Direct-Access     ATA      WDC WD10EZEX-08M 01.0 PQ: 0 ANSI: 5
[   16.170463] sd 1:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/931 GiB)
[   16.176842] sd 1:0:0:0: [sda] 4096-byte physical blocks
[   16.181339] sd 1:0:0:0: [sda] Write Protect is off
[   16.185364] sd 1:0:0:0: Attached scsi generic sg0 type 0
[   16.185393] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[   16.212599]  sda: unknown partition table
[   16.216342] sd 1:0:0:0: [sda] Attached SCSI disk
```

With partition created (in this example there are 4 primary partitions, one for storing file system, one for swap and additional two partitions.

```
[ 4570.941363] ata1: SATA link down (SStatus 0 SControl 300)
[ 4571.108000] ata2: SATA link up 6.0 Gbps (SStatus 133 SControl 300)
[ 4571.113604] ata2.00: ATA-9: WDC WD10EZEX-08M2NA0, 01.01A01, max UDMA/100
[ 4571.119188] ata2.00: 1953525168 sectors, multi 0: LBA48 NCQ (depth 31/32), AA
[ 4571.125625] ata2.00: configured for UDMA/100
[ 4571.129313] scsi 1:0:0:0: Direct-Access     ATA      WDC WD10EZEX-08M 01.0 PQ: 0 ANSI: 5
[ 4571.136422] sd 1:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/931 GiB)
[ 4571.136550] sd 1:0:0:0: Attached scsi generic sg0 type 0
[ 4571.147216] sd 1:0:0:0: [sda] 4096-byte physical blocks
[ 4571.151686] sd 1:0:0:0: [sda] Write Protect is off
[ 4571.155726] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[ 4571.179058]  sda: sda1 sda2
```

When booted with Filesystem on SATA HDD with ext4 file format.

Command line parameters for SATA rootfs.

```
[    0.000000] Kernel command line: console=ttyS0,115200n8 rootwait=1 ro root=/dev/sda1
[    0.000000] PID hash table entries: 4096 (order: 2, 16384 bytes)
```

Log showing mounting for ext4 FS on partition 1 formatted with ext4 file format

```
[  433.556931] EXT3-fs (sda1): error: couldn't mount because of unsupported optional features (240)
[  433.564563] EXT4-fs (sda1): couldn't mount as ext2 due to feature incompatibilities
[  433.578464] EXT4-fs (sda1): INFO: recovery required on readonly filesystem
[  433.584242] EXT4-fs (sda1): write access will be enabled during recovery
[  433.664817] EXT4-fs (sda1): recovery complete
[  433.672994] EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
[  433.679391] VFS: Mounted root (ext4 filesystem) readonly on device 8:1.
```

Port 0 is known to work on customer boards using SATA interface. Internally verified using microTCA AMC chassis and using intel's e1000e NIC card on port 0. The driver for e1000e is supported by intel on its website and IS NOT supported by TI.

There are few dts configuration parameters available to customize the driver.See Documentation/devicetree/bindings/pci/pcie-keystone.txt in Linux tree for DT bindings information.

Procedure to boot Linux with FS on harddisk

Boot Linux kernel on K2E EVM using NFS file system or Ramfs and using rootfs provided in the SDK. Make sure SATA HDD is connected to EVM as explained above and SATA EP is detected during boot up. This example uses a 1TB HDD and create two partition. First partition is for filesystem and is 510GB and second is for swap and is 256MB.

1. create partition with fdisk

First step is to create 2 partitions using fdisk command. At Linux console type the following commands

```
root@keystone-evm:~# fdisk /dev/sda
Welcome to fdisk (util-linux 2.21.2).
```

```
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

```
Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x9b51b66e.
```

```
The device presents a logical sector size that is smaller than
the physical sector size. Aligning to a physical sector (or optimal
I/O) size boundary is recommended, or performance may be impacted.
```

```
Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)
```

```
Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-1953525167, default 2048): 2048
Last sector, +sectors or +size{K,M,G} (2048-1953525167, default 1953525167): +510G
Partition 1 of type Linux and of size 510 GiB is set
```

```
Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First sector (1069549568-1953525167, default 1069549568):
Using default value 1069549568
Last sector, +sectors or +size{K,M,G} (1069549568-1953525167, default 1953525167): +256M
Partition 2 of type Linux and of size 256 MiB is set
```

```
Command (m for help): p
```

```
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x9b51b66e
```

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1069549567   534773760   83  Linux
/dev/sda2      1069549568  1070073855      262144   83  Linux
```

```
Command (m for help): p
```

```
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x9b51b66e
```

```
  Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1069549567   534773760   83  Linux
/dev/sda2      1069549568  1070073855      262144   83  Linux
```

```
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): L
```

```
 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden C:  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi eb  BeOS fs
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         ee  GPT
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ef  EFI (FAT-12/16/
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        f0  Linux/PA-RISC b
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f1  SpeedStor
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f4  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f2  DOS secondary
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      fb  VMware VMFS
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fc  VMware VMKCORE
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fd  Linux raid auto
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fe  LANstep
1c  Hidden W95 FAT3 75  PC/IX           be  Solaris boot    ff  BBT
1e  Hidden W95 FAT1 80  Old Minix
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap / Solaris)
```

```
Command (m for help): p
```

```
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x9b51b66e
```

```
  Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1069549567   534773760   83  Linux
/dev/sda2      1069549568  1070073855      262144   82  Linux swap / Solaris
```

2. Format partitions

```
root@keystone-evm:~# mkfs.ext4 /dev/sda1
mke2fs 1.42.1 (17-Feb-2012)
Filesystem label=
```

```
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
33423360 inodes, 133693440 blocks
6684672 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=0
4080 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
       32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
       4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
       102400000
```

```
Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

```
root@keystone-evm:~# ls -ltr /dev/sda*
brw-rw----    1 root     disk        8,   2 Sep 21 14:37 /dev/sda2
brw-rw----    1 root     disk        8,   0 Sep 21 14:37 /dev/sda
brw-rw----    1 root     disk        8,   1 Sep 21 14:40 /dev/sda1
```

3. Copy filesystem to rootfs

This procedure assumes the cpio file for SDK filesystem is available on the NFS or ramfs.

```
>mkdir /mnt/test
>mount -t ext4 /dev/sda1 /mnt/test
>cd /mnt/test
>cpio -i -v </<rootfs>.cpio
>cd /
>umount /mnt/test
Where rootfs.cpio is the cpio file for the SDK fileystem.
```

4. Booting with FS on harddisk

Once the harddisk is formatted and has a rootfs installed, following procedure can be used to boot Linux kernel using this rootfs.

Boot EVM to u-boot prompt. Add following env variables to u-boot environment :-

```
K2E EVM # setenv boot hdd
K2E EVM # setenv get_fdt_hdd 'dhcp ${addr_fdt} ${tftp_root}/${name_fdt}'
K2E EVM # setenv get_kern_hdd 'dhcp ${addr_kern} ${tftp_root}/${name_kern}'
K2E EVM # setenv get_mon_hdd 'dhcp ${addr_mon} ${tftp_root}/${name_mon}'
K2E EVM # setenv init_hdd 'run set_fs_none  args_all  args_hdd'
K2E EVM # setenv args_hdd 'setenv bootargs ${bootargs} ro root=/dev/sda1'
K2E EVM # saveenv
```

Now type boot command and boot to Linux. The above steps can be skipped once u-boot implements these env variables by default which is expected to be supported in the future.

AER Driver

AER driver is used for PCIE error handling and recovery. AER driver is attached to RC Port. AER driver requires following kernel command line parameters in bootargs

```
pcie_ports=native pcie_aer=nomsi
```

AER driver is tested by simulating a Unsupported Request. To do so, the EP's BAR is corrupted. For the EP driver to send error responses to RC, SERR bit in PCI_COMMAND must be enabled. Also must enable AER error reporting using pci_enable_pcie_error_reporting() API. For example to enable AHCI driver to enable error reporting to RC port, following code (shows diff) was added to drivers/ata/ahci.c in ahci_init_one()

```
#define DRV_VERSION    "3.0"
+extern int pci_enable_pcie_error_reporting(struct pci_dev *dev);
```

```
....
```

```
ahci_init_one(struct pci_dev *pdev, const struct pci_device_id *ent)
.....
```

```
       int ahci_pci_bar = AHCI_PCI_BAR_STANDARD;
+      u16 pci_cmd;
```

```
....
```

```
+       pci_read_config_word(pdev, PCI_COMMAND, &pci_cmd);
+       pci_cmd |= PCI_COMMAND_SERR;
+       pci_write_config_word(pdev, PCI_COMMAND, pci_cmd);
+       pci_enable_pcie_error_reporting(pdev);
        return ata_host_activate(host, pdev->irq, ahci_interrupt,
                 IRQF_SHARED, &ahci_sht);
```

To simulate Unsupported request error, corrupt the memory BAR at EP's config space. For example, write 0 to BAR offset 0x24 as below (dump should show an address in the 0x5xxxxxx range at 0x21802024 assuming CFG_SETUP is pointing to EP's config space) :-

```
>devmem2 0x21802024 w 0x0
```

The above assumes the CFG_SETUP register is pointing to bus 1, dev 0, function 0. If not, read the value at offset 0x8 and update it using dev2mem command. Refer the device UG for details of the register.

For AHCI case, try writing to the disk using following command after corrupting the BAR

```
dd if=/dev/zero of=/media/sda1/test-file.bin bs=1K count=1024
```

The command will time out. Once this is timedout, dmesg command will show something like below for unsupported error received from EP at the RC's port.

```
[  342.828683] pcieport 0000:00:00.0: AER: Multiple Uncorrected (Non-Fatal)
   error received: id=0100
[  342.837572] ahci 0000:01:00.0: PCIe Bus Error: severity=Uncorrected (Non-
   Fatal), type=Transaction Layer, id=0100(Requester ID)
[  342.848916] ahci 0000:01:00.0:   device [1b4b:9125] error
   status/mask=00100000/00000000
[  342.848920] ahci 0000:01:00.0:    [20] Unsupported Request    (First)
[  342.848927] ahci 0000:01:00.0:   TLP Header: 40003001 0000000f 50000134
   00000000
[  342.848945] ahci 0000:01:00.0: broadcast error_detected message
```

```
root@keystone-evm:~# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
 29:          0          0          0          0       GIC  arch_timer
 30:      51241      51511      51736      51561       GIC  arch_timer
 70:          1          0          0          0       GIC  pcie-error-irq, aerdrv
 80:      10858          0          0          0       GIC  hwqueue-8704
 81:          0          0          0          0       GIC  hwqueue-8705
 82:         16          0          0          0       GIC  hwqueue-8706
 83:          0          0          0          0       GIC  hwqueue-8707
 84:          0          0          0          0       GIC  hwqueue-8708[
```

The above shows one error interrupt received by RC port.

More information on AER driver can be seen at Documentation/PCI/pcieaer-howto.txt in Linux source tree.

DMA Coherency

Multicore Cortex-A15 supports cache coherency. So, if one core writes to the memory page and that page has a sharable attribute, corresponding cache entries of other cores will be updated. If SoC doesn't support DMA coherency and external DMA master performs a write transaction L1 and L2 caches will not be updated. So, the software has to invalidate corresponding cache lines. Keystone 2 SoC supports DMA coherency by hardware. MSMC has a special snooping mechanism to monitor bus transactions and communicates to Tetris to update caches. That eliminates necessity to invalidate cache from software. Several conditions must be met in order to support DMA coherency:

- MMU is enabled
- LPAE is enabled
- Linux runs in DDR3A 0x800000000 address space
- US bit in MSMC MPAX Segment registers has to be cleared to enable page sharing and coherency actions. Please read Multicore Shared Memory Controller (http://www.ti.com/lit/ug/spruhj6/spruhj6.pdf) for details.
- Dma-coherent property is set for appropriate node in the DTS.

The MCSDK release meets those requirements because:

- Linux always runs with MMU enabled.
- Kernel has LPAE enabled by default in this release.
- U-boot in arch_cpu_init() calls share_all_segments() functions for ARM and NetCP MPAXes, which clears US bits.
- The default mem_lpae u-boot enviroment variable is set to "1" which makes kernel to work at 0x800000000 DDR3A address space.
- The "dma-coherent" property is set for "pktdma" and "netcp" nodes of tci6638_evm.dts

RT Preempt patch

RT Preempt patch is supported in Linux. The master-rt branch in the Linux repo maintains a patched kernel. keystone2_fullrt_defconfig is used to configure the RT Preempt patched kernel. More details on the RT Preempt patch can be obtained from [4] (https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO#About_the_RT-Preempt_Patch)

Capture kernel crash dump using kexec

When there is a kernel panic, and the system goes down there is a need to capture the coredump before resetting the system. This coredump could be later analyzed to debug the kernel panic and for other purposes. To achieve this with Keystone II devices, a few user space utilities were added as part of the file system. The following sections provides details on what these utilities are and how they were used to enable this feature.

Kexec is a patch to the Linux kernel that allows you to boot directly to a new kernel from the currently running one. In the boot sequence, kexec skips the entire bootloader stage (the first part) and directly jumps into the kernel that we want to boot to. There is no hardware reset, no firmware operation, and no bootloader involved.

Note: kernel crash dump is currently supported only on K2HK EVM for this release

Kexec/Kdump Terminology

Some terms that will be used later on in this section are explained here:

Main-kernel: The kernel that is being used for regular use by the system.
Crash-kernel: The kernel that will be used for bringing up the system at the time of a kernel crash.

Crash-kernel-device tree: The device tree file that will be used by the crash kernel at the time of boot up after a panic from main kernel.

Kexec/Kdump User Space tools

Kexec-tools provides the /sbin/kexec binary that facilitates a new kernel to boot using the kernel's kexec feature either on a normal or a panic reboot. This package contains the /sbin/kexec binary and ancillary utilities that together form the userspace component of the kernel's kexec feature.

We use this tool only at the time of a kernel panic, but like the description says this can be used even to load a different kernel at a particular point in time.

User space kexec tools (version used is v2.0.7) are available in gitweb at: Kexec Git web (http://git.kernel.org/?p=utils/kernel/kexec/kexec-tools.git;a=summary)

We also need to apply a [[patch][5] (http://arago-project.org/git/projects/meta-mcsdk.git?p=projects/meta-mcsdk.git;a=blob;f=meta-arago-extras/recipes-kernel/kexec/kexec-tools-2.0.7/0001-kexec-ks2-use-KS2-aliased-addresses-on.patch;h=6f6eef1fbac4deb35da9b5dbec63594de55a4c4c;hb=refs/heads/master)] on kexec tools v2.0.7 to resolve the address aliasing issue. Keystone II device doesn't have physical memory at the first 4GB address range. In order to access the memory at boot time it aliases two GB of the memory to the that range. So when kexec creates elf header for the crash kernel, it has to convert Keystone II physical addresses to the corresponding aliased ones.

Kexec Usage

To use kexec to load a different kernel at the time of panic, do this: kexec -p <path-to-zImage> --command-line=<boot command line> --dtb=<path-to-device-tree-file>

The -p option indicates that this rule will be applicable at the time of a panic only. The path to zImage is the zImage that will be loaded at the time of a panic, this is called crashkernel. The path to device tree is the device tree file that this crashkernel will use for boot up. The boot command line will provide the command line argument for the new crashkernel to come up.

More options for kexec can be seen by doing: "kexec --help" from the shell prompt.

Reserving size of crashkernel

For kexec-tools to work, a separate area of memory will need to be used to load the crashkernel. This is done by adding "crashkernel=33M@0x810000000" to the bootargs of the main kernel's bootargs. If this is not done, the "kexec -p" command descibed previously will not work and will return an error saying crashkernel size is not allocated.

/proc/vmcore

When the main kernel panics, the crashkernel is triggered and when the crashkernel comes up, the /proc/vmcore entry will show the exact state of the previous kernel's view of the system. Capturing this file is needed to analyze the previous kernel panic. But this file will be really huge (because this is in effect, the previous kernel's view of the system). So there are some techniques done to get the exact coredump file that we want which are discussed in the next sections.

makedumpfile

With kexec/kdump, the memory image of the first kernel (called "main kernel") can be taken as /proc/vmcore while the second kernel (called "crash kernel") is running. makedumpfile makes a small DUMPFILE from the /proc/vmcore by compressing dump data or by excluding unnecessary pages for analysis, or both. makedumpfile needs the first kernel's debug information, so that it can distinguish unnecessary pages by analyzing how the first kernel uses the memory.

More information on makedumpfile can be found at: makedump file (http://linux.die.net/man/8/makedumpfile)

There is one particular usage of makedumpfile that we will be employing to capture relevant detail:

```
   makedumpfile -E -d 31 /proc/vmcore/ coredump
   the -E option means the output file coredump will be in ELF format. (The default is in "crash" format, which needs "crash" utility)
   -d 31 means all zero pages, cache pages, cache private, user and free pages are removed when generating the coredump file. This is needed otherwise the coredump file will
become really huge.
```

makedumpfile option Description for "-d"
1 Zero pages
2 Cache pages
4 Cache private
8 User pages
16 Free pages

Refer for more information: makedumpfile options (https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/ch-kdump.html)

Crashkernel and crash kernel device-tree

Our kernel uImage file is relocatable and automatically relocates to 0x80008000, so to be used for crashkernel the zImage is preferred and tested.

At the time of kernel panic, the idea is that none of the hardware modules are reliable - because there may be some data traffic happening, and file system may be also be corrupted. So the idea is to bring up a crashkernel that does not have any of the hw modules enabled, and crash kernel will use a new filesystem. The "k2hk_evm_recovery_defconfig" has all TI_KEYSTONE disabled, networking disabled etc. Correspondingly we need a device tree file also to be used by the crashkernel. This device tree file need to provide the bootargs for the crashkernel, and listthe "elfcorehdr=xxx" for the crashkernel to generate the /proc/vmcore entry.

This device tree source file is under <linux-repo>/arch/arm/boot/dts/k2hk-evm-recovery.dts.

The crashkernel can be built from our Linux kernel repo, by doing:
make keystone2_recovery_defconfig
make
make <device>-evm-recovery.dtb

The output zImage is in <linux-repo>/arch/arm/boot

Normally when the kernel is booted by the u-boot, the u-boot takes care to reserve free memory space in the DTB, which is required for the kernel. User has to add the free space manually using dtc complier:

dtc -I dtb -O dtb -p 2048 -o <device>-evm-recovery.dtb arch/arm/boot/dts/k2hk-evm-recovery.dtb

The output DTB is in <linux-repo>/

Note: <device> is k2hk, k2l or k2e

Init-script to run in the crash kernel's filesystem

The crash kernel will use a filesystem and the main expectation from this filesystem is to extract the vmcore information to a elf coredump. For this purpose, an init-script is written which gets executed automatically from the recovery-filesystem. It looks like below:

```
#!/bin/sh
if [ -f /proc/vmcore ] ; then
  mkdir -p /mnt/boot2
  mount -o rw,sync -t ubifs /dev/ubi0_2 /mnt/boot2
  if [ $? -eq 0 ] ; then
     if [ -f /usr/bin/makedumpfile ] ; then
       makedumpfile -E -d 31 /proc/vmcore /mnt/boot2/home/root/coredump.elf
       gzip -c /mnt/boot2/home/root/coredump.elf > /mnt/boot2/home/root/coredump.elf.gz
       rm -rf /mnt/boot2/home/root/coredump.elf
       sync
       reboot
     else
       echo "makedumpfile not found"
       exit 1
     fi
  else
    echo "mount unsuccessful"
    exit 1
  fi
fi
exit 0
```

Note: This script is part of the rootfs-recovery filesystem by default under: /etc/rc3.d/S09recoveryfs

How to enable crashdump kernel with MCSDK on K2HK-EVM

The default root-filesystem of SC-MCSDK contains the kexec and kdump user space utilities under /usr/sbin. The crashkernel and crash kernel's device tree binary file is pre-built and located under /usr/bin/crashdump. This kernel is built based on the "k2hk_evm_recovery_defconfig" located under arch/arm/configs of the linux repo. This device tree file is built based on the "k2hk-evm-recovery.dts" located under <linux-repo>/arch/arm/boot/dts/.

To verify the kexec way of loading a crashkernel and capturing main kernel panic information, use steps below:

From a UBI based NAND filesystem, do the following:

```
Boot the system with the following text added to the args_all line in u-boot:

"crashkernel=33M@0x810000000"

Boot the system.

cd /usr/bin/crashdump
```

For k2hk or k2e:

```
kexec -p zImage --command-line="console=ttyS0,115200n8 earlyprintk rootwait=1 maxcpus=1 rootfstype=ubifs root=ubi0:rootfs-recovery rootflags=sync rw ubi.mtd=2,2048" --dtb=
<device>-evm-recovery.dtb
```

For k2l:

```
kexec -p zImage --command-line="console=ttyS0,115200n8 earlyprintk rootwait=1 maxcpus=1 rootfstype=ubifs root=ubi0:rootfs-recovery rootflags=sync rw ubi.mtd=2,4096" --dtb=
<device>-evm-recovery.dtb
```

Note: <device> is k2hk, k2l or k2e.

```
At this time, the main kernel is configured to use the zImage as the crash kernel.
Now invoke a panic, by manually building a simple panic kernel module.
```

A sample panic.c which can be built as kernel module is shown below:

```
/*
 * File name: panic.c
 */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sort.h>
#include <linux/bsearch.h>
#include <linux/xfrm.h>
#include <net/net_namespace.h>
#include <linux/list.h>
#include <linux/hash.h>
MODULE_LICENSE("GPL v2");
static int __init panic_init(void)
{
      printk("calling panic()\n");
      panic("panic has been called");
      return 0;
}
module_init(panic_init);
```

Now do:

```
insmod panic.ko
```

This will reboot the board and:
1. Come up to the new dump-kernel/recovery-fs (because the crash kernel device tree file uses the recovery filesystem)
2. Automatically copy the elf-core-dump (the initscripts of recovery filesystem checks for /proc/vmcore entry and generates the coredump elf)
3. Reboots back to the original kernel/fs. (Again, the initscript of recovery fs does a reboot after copying the coredump)

Now: from /home/root directory, check if there is a coredump.elf.gz. This can be transferred to the linux pc, and analyzed with gdb against a vmlinux file.

```
arm-linux-gnueabihf-gdb <vmlinux-file> --core=coredump.elf
```

Graceful Power shutdown

This is a sample implementation to support graceful power shutdown. To have a complete implementation, it is assumed that user space daemon listen to the power button event and initiate shutdown.

Smart Reflex Class 0

Please refer to Smart Reflex Class 0 software User's Guide [6] (http://git.ti.com/cgit/cgit.cgi/keystone-linux/srss-tc.git/plain/docs/srss_c0_ug.pdf) for details.

EVM Setup

This is the sample configuration used for testing the graceful power shutdown use case. The DTS bindings may be altered for the individual GPIO pins used for the target board and also others as needed for the specific usecase.

```
      GPIO pin 1 (power button)
BMC  -----------------> SOC
```

```
      GPIO pin 2 (shutdown complete - power off)
BMC  <----------------- SOC
```

Tests

Used BMC version - 1.0.2.5 Use Linux kernel image with patches applied Boot up the EVM

test power button event reception at the user space

---

step 1. type the following command at the BMC to enable hwdbg

```
[00:00:07]  BMC>hwdbg cmd gpio show
[00:00:07]  Executing command "hwdbg"
[00:00:07]  Enabling command: gpio
```

step 2. type the following command at the Linux console to wait for the power button event.

```
root@keystone-evm:~# evtest /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio_keys.7"
Supported events:
 Event type 0 (Sync)
 Event type 1 (Key)
 Event code 116 (Power)
Testing ... (interrupt to exit)
```

step 3. type the following command at the BMC to toggle GPIO pin 1 to high.

```
BMC>gpio xa  XXXX_XXXXXX_XX1X
[19:26:21]  Executing command "gpio"
[19:26:21]  Writing to GPIO expander A...
[19:26:21]  Inputs: 0x0000
High Outputs: 0x0002
Low Outputs: 0x0000
[19:26:21]  Finished setting GPIO expander A...
```

Following log will display at the linux console showing the Power button event when GPIO pin is toggled to high

```
Event: time 1373363555.593249, type 1 (Key), code 116 (Power), value 1
Event: time 1373363555.593249, ---------- Report Sync ------------
```

test power off event from SoC to BMC

---

step 1. type the following command to at the BMC to display the current GPIO pin 2 state

```
[19:37:10]  BMC>gpio xa
[19:37:10]  Executing command "gpio"
[19:37:10]  Reading from GPIO expander A...
[19:37:10]  Inputs:  0000_0000_0000_0001
[19:37:10]  Outputs: XXXX_XXXX_XXXX_XXXX
```

step 2. Initiate shutdown command from user space by typing following command at the Linux console

```
root@keystone-evm:~# shutdown -P -h "now"
```

```
INIT: Sending processes the TERM signalm (ttyS0) (Tue Jul  9 10:02:20 2013):
INIT:Stopping telnet daemon.
Stopping tiipclad daemon.
Stopping syslogd/klogd: stopped syslogd (pid 1586)
stopped klogd (pid 1589)
done
Stopping thttpd.
NOT deconfiguring network interfaces: / is an NFS mount
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
[  460.561642] Power down.
```

step 3. type the following command to at the BMC to display the current GPIO pin 2 state again

```
[19:37:12]  BMC>gpio xa
[19:37:12]  Executing command "gpio"
[19:37:12]  Reading from GPIO expander A...
[19:37:12]  Inputs:  0000_0000_0000_0101
[19:37:12]  Outputs: XXXX_XXXX_XXXX_XXXX
```

Note that pin2 state is toggled to 1 (in step.1 it was showing 0)

configuration

The feature is enabled by default in the default image shipped with release. To get this working on a board, user need to identify the GPIO pins used and add following bindings to the DTS file

```
            gpio_poweroff {
                    compatible = "gpio-poweroff";
                    /*
                     * Change this as needed in the target board to match the
                     * pin used for signaling Power down event from the SoC.
                     * It is assumed that external controller reads this pin
                     * and remove power to the SoC. On EVM, pin 4 output
                     * can be read from BMC to test this.
                     */
                    gpios = <&gpio0 4 0>;
            };
```

```
            gpio_keys {
                    compatible = "gpio-keys";
                    #address-cells = <1>;
                    #size-cells = <0>;
```

```
                    button@1 {
                            label = "Power button";
                            /* this is the key power button defined in Linux input.h */
                            linux,code = <116>;
                            /* EV_KEY */
                            linux,input-type = <1>;
                            /*
                             * On EVM use GPIO pin 3 since it is also connected to
                             * BMC to generate a shutdown event to ARM. Change this
                             * as needed in the target board.
                             */
                            gpios = <&gpio0 3 0>;
                    };
            };
```

Yocto

MCSDK uses Yocto projet to build the Linux kernel, U-boot, user space components, and root filesystem.

Prerequisite

To build yocto in an ubuntu Linux machine, first get a library for 32-bit compilation:

```
sudo apt-get -y install gcc-multilib libc6-i386
```

Next, the following tools should be installed:

```
sudo apt-get -y install diffstat texi2html texinfo subversion chrpath build-essential subversion ccache sed wget cvs coreutils unzip texinfo docbook-utils gawk help2man file g++ bison flex htmldoc chrpath libxext-dev xserver-xorg-dev doxygen socat uboot-mkimage git
```

Note: The apt-get steps only needs to be done once.

Configuration

The following steps set up a default configuration that may be customized as needed:

NOTE: The first step is changing for the MCSDK 3.0.4 POST GA release. If using release earlier releases use the old procedure under: oe-layersetup for older releases.

- Clone the oe-layersetup.git (git://arago-project.org/git/projects/oe-layersetup.git) from Arago project

```
$ git clone http://arago-project.org/git/projects/oe-layersetup.git
$ cd oe-layersetup
```

oe-layersetup for older releases

- Clone the oe-layersetup-mcsdk.git (http://arago-project.org/git/people/hzhang/oe-layersetup-mcsdk.git) from Arago project

```
$ git clone http://arago-project.org/git/people/hzhang/oe-layersetup-mcsdk.git mcsdk
$ cd mcsdk
```

- Choose the configuration file based on the release used.

```
The configuration file for the mcsdk release is kept at configs/mcsdk/mcsdk-<version>-config.txt
The latest mcsdk release configuration file is configs/mcsdk/mcsdk-03.01.03.06-config.txt
Note: During development process a different configuration file may be used.
```

- Run the setup script to configure oe-core layers and builds (Note to use the right config file here)

```
$ ./oe-layertool-setup.sh -f configs/mcsdk/mcsdk-03.01.03.06-config.txt
```

```
Note: For mcsdk 3.1.4.7: In addition to the config script, need to run the following steps

$ cd sources/oe-core
$ git cherry-pick 3fa24eee41c26fecd5e4f680082288ec772d2de9
$ cd -
```

Build

This section describes how the linux rootfs filesystem can be built.
This section assumes that the Linaro toolchain for ARM is installed and environment variables CROSS_COMPILE, ARCH are set up as per instructions given in section Toolchain Installation Linaro_toolchain (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Tools#Linaro_toolchain).

export CROSS_COMPILE=arm-linux-gnueabihf-
export ARCH=arm
PATH=<path to installed toolchain>/bin:$PATH

To build the linux rootfs filesystem, one can either use the local snapshot of the sources in MCSDK release, or fetch the latest source packages at external URLs.

Using the snapshot of the source packages in MCSDK release

Using the snapshot of the arago source packages can avoid fetch errors during the build when external URLs become unavailable. To use the snapshot of sources distributed with MCSDK release, for example, see mcsdk-arago-source (http://downloads.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/exports/mcsdk-3_01_03_06.arago.src.tar.gz). Once this package is downloaded, the content can be un-tarred under the directory $(INSTALL_DIR)/downloads. With this, the build will use local source packages in the downloads directory instead of fetching the latest from external URLs.

- un-tar the arago source tarball

```
$ cd oe-layersetup
$ ls

    build configs oe-layertool-setup.sh sample-files sources

$ tar xzvf $(DOWNLOAD_DIR)/mcsdk-3_##_##_##.arago.src.tar.gz
$ ls

    build configs downloads oe-layertool-setup.sh sample-files sources
```

- Then, follow all the steps of using the latest source packages on internel URLs below to build

Note: for MCSDK 3.0.4, the build version is 3_00_04_18; for MCSDK 3.1.0, the build version is 3_01_00_03; for MCSDK 3.1.1, the build version is 3_01_01_04; for MCSDK 3.1.2, the build version is 3_01_02_05; for MCSDK 3.1.3, the build version is 3_01_03_06.

Build procedure

- Set up the environment variables and start the build using the configuration file

```
$ cd build
$ source conf/setenv
$ MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake tisdk-server-rootfs-image
```

For releases older than MCSDK 3.0.4, ARAGO_BRAND=mcsdk is not required. And also use <soc> as "keystone".

For the releases MCSDK 3.0.4 and after use: <soc> is "k2hk", "k2l" or "k2e";

Once build is complete, the images for the kernel and file system can be located at

$(INSTALL_DIR)/build/arago-tmp-external-linaro-toolchain/deploy/images

- If needed, edit the bitbake configuration file conf/local.conf to customize your bitbake build. In particular, enabling parallel task execution substantially speeds up the Arago build process by fetching package sources in parallel with other build steps. In order to do so, please enable and edit the following parameter in your bitbake configuration BB_NUMBER_THREADS = "4"

Note: The BB_NUMBER_THREADS definition is not the number of CPUs on your SMP machine. The value of 4 appears to work quite well on a single core system, and may be adjusted upwards on SMP systems.

Note: If you are developing applications outside Yocto, you can point the your build's sysroot to the sysroot, Yocto has generated using above steps. You can add the following to your build option *--sysroot=$(INSTALL_DIR)/build/arago-tmp-external-linaro-toolchain/sysroots/keystone-evm* to pick up the sysroot, the Yocto build has created.

Note: For MCSDK 3.00.04.18 release, in the file: ./sources/meta-mcsdk/recipes-core/rhino/rhino_1.7r4.bbappend file: The SRC_URI needs to be changed from: ftp://ftp.freebsd.org/pub/FreeBSD/ports/distfiles/rhino/rhino1_7R4.zip to http://distcache.freebsd.org/ports-distfiles/rhino/rhino1_7R4.zip

Building other components in Yocto

There are other open source components for which are recipes are available in the yocto build system. And customers may choose to build them and add it to their file system.

Any open source component for the platform, if available and compatible with the platform, can be built for the using the following bitbake command similar the build procedure for the filesystem.

MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake <name_of_recipe_without_version and file extension>

For releases older than MCSDK 3.0.4, ARAGO_BRAND=mcsdk is not required. And also use <soc> as "keystone".

For the releases MCSDK 3.0.4 and after use: <soc> is "k2hk", "k2l" or "k2e";

(e.g) MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake sudo

This depending on the recipe will create the ipks under the appropriate directory structure under

$(INSTALL_DIR)/build/arago-tmp-external-linaro-toolchain/deploy/ipk

Updating a user space component in MCSDK filesystem

The following procedure can be used to update a user space component already available in MCSDK filesystem.
Prerequisite: Follow the procedure to create the build setup for Yocto build for the base MCSDK release.

Step1: Update the component recipe

Locate the component recipe under the different YOCTO layers/git repositories cloned under oe-layersetup/sources/*.
And modify locally or patch the recipe.
(And if needed create a patch to keep it for reference, to help recreate the build)

Step 2: Build the component

(Assuming the arm tools are setup based on yocto build instructions)

```
$ cd build
$ . conf/setenv
$ MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake <component_recipe_name>
```

Eg.

```
MACHINE=k2hk-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake ti-pa-bin
```

(Note with this build complete, installable user space packages for the module will be created under: oe-layersetup/build/arago-tmp-external-linaro-toolchain/deploy/ipk/cortexa15hf-vfp-neon-3.8 with name <module-name>*.ipk.
e.g ti-pa-bin_03.00.00.10-r0_cortexa15hf-vfp-neon-3.8.ipk : This is the main package to be used with filesystem.
ti-pa-bin-dbg_03.00.00.10-r0_cortexa15hf-vfp-neon-3.8.ipk : This is debug package (Normally not used).
ti-pa-bin-dev_03.00.00.10-r0_cortexa15hf-vfp-neon-3.8.ipk : This is dev package to be used with devkit.
)

Step 3: Add package to filesystem

There are two ways this can be added to the filesystem.
On the host machine:
- Untar the filesystem in a host directory. (e.g)

```
mkdir rootfs
cd rootfs
sudo tar xzf  ../tisdk-rootfs-<k2e|k2h|k2l>-evm.tar.gz
```

- Install the package into the filesystem.

```
dpkg -x <dirname>/<component_recipe_name>_*.ipk <dirname>/rootfs
```

(e.g)

```
dpkg -x  ~/ti-pa-bin*.ipk ~/rootfs
```

- Repackage the file system

```
sudo tar czvf  . ../ tisdk-rootfs-<k2e|k2h|k2l>-evm-modified.tar.gz
```

(Now this filesytem can be used to create the UBI image etc).
On the target:
- Tftp the *.ipks into the filesytem. And use the following command to install it

```
opkg install <component_recipe_name>_*.ipk
```

e.g:

```
opkg install ti-pa-bin_*.ipk
```

FAQ

1. Error in coying toolchain files:

```
| DEBUG: Executing shell function do_install
| cp: cannot stat '/opt/linaro-3.1.0.3/sysroots/i686-arago-linux/usr/arm-linux-gnueabihf/libc/usr/share/*': No such file or directory
```

Make sure the default shell is bash. If not, please use the following command:
(to choose bash instead of dash)

```
sudo dpkg-reconfigure dash
```

Flattened Device Tree

The Device Tree is a data structure for describing hardware. Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time. The device tree is used both by Open Firmware, and in the standalone Flattened Device Tree (FDT) form. For more details refer [[7] (http://www.devicetree.org/Main_Page)]

creating dtb image

Kernel tree has device tree compiler and can be build as part of the kernel build process. DTB can also be built using the dtc compiler part of the rootfs that is shipped with the kernel. Do the following to copy the dts files to rootfs (either on the Host nfs root directory or tftp to the EVM from Linux console) and build dtb

```
git clone git://git.ti.com/keystone-linux/linux.git linux-keystone
cd linux-keystone
git reset --hard <release tag>
```

The release tag is obtained from the Release notes. Copy arch/arm/boot/dts/k2hk-evm.dts and arch/arm/boot/dts/skeleton.dtsi to the rootfs (either on Host nfs root directory if using nfs rootfs or rootfs on NAND on EVM if using rootfs on NAND)

Once the Linux booted up with the released image, do the following:-

```
cd <path where the dts files are copied>
dtc -I dts -O dtb -o uImage-k2hk-evm.dtb k2hk-evm.dts
```

Copy the uImage-k2hk-evm.dtb to the tftp server or boot volume of the UBI depending on the rootfs used and reboot the EVM.

UBI/UBIFS

UBI

UBI (Latin: "where?") stands for "Unsorted Block Images". It is a volume management system for raw flash devices which manages multiple logical volumes on a single physical flash device and spreads the I/O load (i.e, wear-leveling) across whole flash chip.

In a sense, UBI may be compared to the Logical Volume Manager (LVM). Whereas LVM maps logical sectors to physical sectors, UBI maps logical eraseblocks to physical eraseblocks. But besides the mapping, UBI implements global wear-leveling and transparent I/O errors handling.

An UBI volume is a set of consecutive logical eraseblocks (LEBs). Each logical eraseblock may be mapped to any physical eraseblock (PEB). This mapping is managed by UBI, it is hidden from users and it is the base mechanism to provide global wear-leveling (along with per-physical eraseblock erase counters and the ability to transparently move data from more worn-out physical eraseblocks to less worn-out ones).

UBI volume size is specified when the volume is created and may later be changed (volumes are dynamically re-sizable). There are user-space tools which may be used to manipulate UBI volumes.

There are 2 types of UBI volumes - dynamic volumes and static volumes. Static volumes are read-only and their contents are protected by CRC-32 checksums, while dynamic volumes are read-write and the upper layers (e.g., a file-system) are responsible for ensuring data integrity.

UBI is aware of bad eraseblocks (e.g., NAND flash may have them) and frees the upper layers from any bad block handling. UBI has a pool of reserved physical eraseblocks, and when a physical eraseblock becomes bad, it transparently substitutes it with a good physical eraseblock. UBI moves good data from the newly appeared bad physical eraseblocks to good ones. The result is that users of UBI volumes do not notice I/O errors as UBI takes care of them.

NAND flashes may have bit-flips which occur on read and write operations. Bit-flips are corrected by ECC checksums, but they may accumulate over time and cause data loss. UBI handles this by moving data from physical eraseblocks which have bit-flips to other physical eraseblocks. This process is called scrubbing. Scrubbing is done transparently in background and is hidden from upper layers.

Here is a short list of the main UBI features:

- UBI provides volumes which may be dynamically created, removed, or re-sized;
- UBI implements wear-leveling across whole flash device (i.e., you may continuously write/erase only one logical eraseblock of an UBI volume, but UBI will spread this to all physical eraseblocks of the flash chip);
- UBI transparently handles bad physical eraseblocks;
- UBI minimizes chances to lose data by means of scrubbing.

For more information on UBI, refer [[8] (http://www.linux-mtd.infradead.org/doc/ubi.html)[UBI]]

UBIFS

UBIFS may be considered as the next generation of the JFFS2 file-system.

JFFS2 file system works on top of MTD devices, but UBIFS works on top of UBI volumes and cannot operate on top of MTD devices. In other words, there are 3 subsystems involved:

MTD subsystem, which provides uniform interface to access flash chips. MTD provides an notion of MTD devices (e.g., /dev/mtd0) which basically represents raw flash UBI subsystem, which is a wear-leveling and volume management system for flash devices. .UBI works on top of MTD devices and provides a notion of UBI volumes; UBI volumes are higher level entities than MTD devices and they are devoid of many unpleasant issues MTD devices have (e.g., wearing and bad blocks); For more information on MTD, refer [[9] (http://www.linux-mtd.infradead.org/doc/general.html)[MTD]]

For more information on UBIFS, refer [[10] (http://www.linux-mtd.infradead.org/doc/ubifs.html)[UBIFS]]

UBIFS User-space tools

UBI user-space tools, as well as other MTD user-space tools, are available from the the following git repository: git://git.infradead.org/mtd-utils.git

The repository contains the following UBI tools:

- ubinfo - provides information about UBI devices and volumes found in the system
- ubiattach - attaches MTD devices (which describe raw flash) to UBI and creates corresponding UBI devices
- ubidetach - detaches MTD devices from UBI devices (the opposite to what ubiattach does)
- ubimkvol - creates UBI volumes on UBI devices
- ubirmvol - removes UBI volumes from UBI devices
- ubiupdatevol - updates UBI volumes. This tool uses the UBI volume update feature which leaves the volume in "corrupted" state if the update was interrupted; additionally, this tool may be used to wipe out UBI volumes.
- ubicrc32 - calculates CRC-32 checksum of a file with the same initial seed as UBI would use
- ubinize - generates UBI images
- ubiformat - formats empty flash, erases flash and preserves erase counters, flashes UBI images to MTD devices
- mtdinfo - reports information about MTD devices found in the system.

All UBI tools support "-h" option and print sufficient usage information.

NAND Layout

The NAND flash in the EVM contains three partitions:-

- bootloader - Contains u-boot
- params - contains env variables
- ubifs - contains following UBI volumes:-
    - boot volume - contains Kernel image (uImage), device tree blob etc,
    - rootfs volume - contains the rootfs which is the primary filesystem

Note: bootloader partition is blank, u-boot is stored on SPI NOR flash.

Compiling UBIFS Tools

The MTD and UBI user-space tools are available from the the following git repository: git://git.infradead.org/mtd-utils.git

Suggest using 1.4.8 of the mtd-utils

For instructions on compiling MTD-utils, refer [[11] (http://processors.wiki.ti.com/index.php/MTD_Utilities#MTD-Utils_Compilation)[MTD-Utils Compilation]]. In the instruction for building mtd-utils, please replace PREFIX with INSTALL_DIR. The makefile doesn't like the use of PREFIX variable and result in build error. This is a work around to fix the build error.

Creating UBIFS file system

For information on how to create a UBIFS image. refer [[12] (http://www.linux-mtd.infradead.org/faq/ubifs.html#L_mkfubifs)]

- mkfs.ubifs

```
mtd-utils# mkfs.ubifs -r filesystem/ -o ubifs.img -F -m 2048 -e 126976 -c 936
```

Where

- -m 2KiB (or 2048). The minimum I/O size of the underlying UBI and MTD devices. In our case, we are running

  the flash with no sub-page writes, so this is a 2KiB page.

- -e 124KiB (or 126976) Erase Block Size: UBI requires 2

  minimum I/O units out of each Physical Erase Block (PEB) for overhead: 1 for maintaining erase count information, and 1 for maintaining the Volume ID information. The PEB size for the Micron flash is 128KiB, so this leads to each Logical Erase Block (LEB) having 124KiB available for data.

- -c 936 The maximum size, in LEBs, of this file system. See calculation below for how this number is determined.
- -r filesystem. Use the contents of the 'filesystem/' directory to generate the initial file system image.
- -o ubifs.img Output file.
- -F parameter is used to set the "fix up free space" flag in the superblock, which forces UBIFS to "fixup" all the free space which it is going to use.

The output of the above command, ubifs.img is fed into the 'ubinize' program to wrap it into a UBI image.

Creating UBI image

The images produced by mkfs.ubifs must be further fed to the ubinize tool to create a UBI image which must be put to the raw flash to be used a UBI partition.

- Create ubinize.cfg file and write the contents into it

```
mtd-utils# vi ubinize.cfg
[ubifs_rootfs_volume] <== Section header
 mode=ubi <== Volume mode (other option is static)
 image=ubifs.img <== Source image
 vol_id=0 <== Volume ID in UBI image
 vol_size=113MiB <== Volume size
 vol_type=dynamic <== Allow for dynamic resize
 vol_name=rootfs <== Volume name
 vol_flags=autoresize <== Autoresize volume at first mount [See calculations below to determine the value
 associated with 'vol_size']
```

Note: There is a sample ubinize.cfg under mcsdk_linux_3_00_00_xx/images directory which is used to create the MCSDK ubi image.

- ubinize

```
mtd-utils# ubi-utils/ubinize -o ubifs.ubi -m 2048 -p 128KiB -s 2048 -O 2048 ubinize.cfg
```

Where:

- -o ubifs.ubi Output file
- -m 2KiB (or 2048) Minimum flash I/O size of 2KiB page
- -p 128KiB Size of the physical eraseblock of the flash this UBI image is created for
- -s 2028 Use a sub page size of 2048 -O 2048 offset if the VID header from start of the physical eraseblock

The output of the above command, 'ubifs.ubi' is the required image.

Calculations

Usable Size Calculation As documented here, UBI reserves a certain amount of space for management and bad PEB handling operations. Specifically:

- 2 PEBs are used to store the UBI volume table
- 1 PEB is reserved for wear-leveling purposes;
- 1 PEB is reserved for the atomic LEB change operation;
- a % of PEBs is reserved for handling bad EBs. The default for NAND is 1%
- UBI stores the erase counter (EC) and volume ID (VID) headers at the beginning of each PEB.
- 1 min I/O unit is required for each of these.

To calculate the full overhead, we need the following values:

```
Symbol          Meaning                                                 value
------          -------                                                 ---------------------------
SP              PEB Size                                                128KiB
SL              LEB Size                                                128KiB - 2 * 2KiB = 124 KiB
P               Total number of PEBs on the MTD device                  126.5MiB / 128KiB = 1012
B               Number of PEBs reserved for bad PEB handling            1% of P = 10.12(round to 10)
O               The overhead related to storing EC and VID headers in bytes,    4KiB
                i.e. O = SP - SL
```

Assume a partition size of 126.5M (We use 1.5M for bootloader and params leaving 126.5M for ubifs partition)

UBI Overhead = (B + 4) * SP + O * (P - B - 4) = (10 + 4) * 128Kib + 4 KiB * (1012 - 9- 4) = 5784 KiB = 45.1875 PEBs (round to 45)

This leaves us with 967 PEBs or 123776 KiB available for user data.

Note that we used "-c 998 " in the above mkfs.ubifs command line to specify the maximum filesystem size, not "-c 967" The reason for this is that mkfs.ubifs operates in terms of LEB size (124 KiB), not PEB size (128Kib). 123776 KiB / 124 Kib

```
= 998.19 (round to 998).
```

Volume size = 123776 KiB (~120 MiB)

Use this caculation method to calculate the size required for each ubifs image going into each of the ubifs volumes on NAND.

A sample ubinize.cfg file is included in the images folder of the release. This was used to create the keystone-evm-ubifs.ubi image. There is also a keystone-evm-boot.ubifs that is part of the release images folder. This was created as follows: Use the mkfs.ubifs and ubinize utilities provided with the release (under bin folder)

```
cd <release_folder>/mcsdk_linux_<version>/images
mkdir boot
cp uImage-keystone-evm.bin boot/
cp uImage-k2hk-evm.dtb boot/
cp skern-keystone-evm.bin boot/
export PATH=<release_folder>/mcsdk_linux_<version>/bin:$PATH
mkfs.ubifs -r boot -F -o keystone-evm-boot.ubifs -m 2048 -e 126976 -c 41
cp keystone-evm-boot.ubifs images/
cd images/
ubinize -o keystone-evm-ubifs.ubi -m 2048 -p 128KiB -s 2048 -O 2048 ubinize.cfg
```

Using UBIFS file system

Preparing NAND partition Kindly erase the NAND partition before using it for UBI file system. The partition can be erased from either u-boot or from Linux.

Follow below steps to erase.

From U-boot,

1. Reset environment variables to default values and set the boot mode to ubi:

```
u-boot# env default -f -a
u-boot# setenv boot ubi
```

Note: On Rev 1.0 or older EVMs, the NAND part used is 128M. So following commands needed to set mtdparts

```
u-boot# setenv mtdparts 'mtdparts=davinci_nand.0:1024k(bootloader)ro,512k(params)
        ro,129536k(ubifs)'
```

2. Set serverip and tftp_root env variables:

```
u-boot# setenv serverip <TFTP server IP address>
u-boot# setenv tftp_root <directory under TFTP root>
u-boot# saveenv
```

3. Create ${tftp_root} directory under TFTP root and copy mcsdk_linux_3_00_00_xx/images/keystone-evm-ubifs.ubi to ${tftp_root} directory

4. TFTP keystone-evm-ubifs.ubi to DDR3 memory and write ubi image from DDR3 to ubifs partition on NAND:

```
u-boot# setenv addr_file 0x82000000
u-boot# nand erase.part ubifs
u-boot# dhcp ${addr_file} ${tftp_root}/keystone-evm-ubifs.ubi
u-boot# nand write ${addr_file} ubifs ${filesize}
```

or use the automated scripts

```
u-boot# run get_ubi_net
u-boot# run burn_ubi
```

Note that the above assumes name_ubi env variable is set to "keystone-evm-ubifs.ubi". If you are using rt version of the ubi image, change name_ubi env variable to match with the rt file name.

Note: the file size of the keystone-evm-ubifs.ubi is printed on console when tftp is completed.

Note: If you do not have a DHCP server on your network, you may assign a static IP using the U-Boot command "setenv ipaddr <ipadd>".

Note: if running on Rev 1.0 or older EVMs, please update dts for correct mtd partition size to account for 128M NAND available as follows:-

Also DTS file needs to be modified for the size of NAND as well as follows:-

```
 nand@2,0 {
     .....
     partition@180000 {
        label = "ubifs";
        reg = <0x180000 0x7e80000>;
     };
 };
```

Rebuild DTB and create a boot ubifs image (keystone-evm-boot.ubifs) using the new DTBas described at the end of the section creating boot ubifs. Then remove and create a new boot volume on UBI from u-boot using following commands

```
u-boot# dhcp ${addr_file} ${tftp_root}/keystone-evm-boot.ubifs
u-boot# ubi part ubifs
u-boot# ubifsmount boot
u-boot# ubi remove boot
u-boot# ubi create boot [size in hex of the ubifs image]
u-boot# ubi write $addr_file} boot [size in hex of the ubifs image]
```

From Linux. Assuming MTD partition 2 needs to be erased and used for UBI file system.

```
root@arago-armv7:~# flash_eraseall /dev/mtd2
```

Flashing UBIFS image to a NAND partition

We can Flash UBIFS image from either Linux Kernel or U-Boot.

Follow steps mentioned here to create an UBIFS image.

From Linux,

Flash the UBI file system image (ubifs.ubi) to MTD partition "X"

ubiformat /dev/mtd<X> -f ubifs.ubi -s 2028 -O 2048

Assuming 2nd mtd partition, we can use the following command to flash the ubifs ubi image to partition 2.

```
#flash_erase /dev/mtd2 0 0
#ubiformat /dev/mtd2 -f keystone-evm-ubifs.ubi -s 2048 -O 2048
```

Using UBIFS image as root file system

Set up the bootargs environment variable as below to use the UBIFS file system image present in a MTD partition:

```
setenv bootargs 'console=ttyS0,115200n8 mem=512M earlyprintk debug rootwait=1 rw ubi.mtd=X,YYYY rootfstype=ubifs
root=ubi0:rootfs rootflags=sync'
```

Where X is the MTD partition number being used for file system and YYYY is the NAND page size. make sure that an UBI file system is flashed into this partition before passing it as a boot partition for Linux.

Assuming 2nd mtd partition and page size of 2048,

```
setenv bootargs 'console=ttyS0,115200n8 mem=512M earlyprintk debug rootwait=1 rw ubi.mtd=2,2048 rootfstype=ubifs
root=ubi0:rootfs rootflags=sync'
```

Updating Boot volume images from Linux kernel

The files in the boot volume may be removed and replaced by new file and EVM may be rebooted using the new images. See below the steps to replace the file in boot volume.

```
root@keystone-evm:~# mkdir /mnt/boot
root@keystone-evm:~# mount -t ubifs ubi0_0 /mnt/boot
```

<syntaxhighlight lang="c"> [1337.657081] UBIFS: mounted UBI device 0, volume 0, name "boot" [1337.663070] UBIFS: file system size: 3936256 bytes (3844 KiB, 3 MiB, 31 LEBs)u [1337.670334] UBIFS: journal size: 1142785 bytes (1116 KiB, 1 MiB, 8 LEBs) [1337.677502] UBIFS: media format: w4/r0 (latest is w4/r0) [1337.683297] UBIFS: default compressor: lzo [1337.687360] UBIFS: reserved for root: 0 bytes (0 KiB) </syntaxhighlight>

```
root@keystone-evm:~# root@keystone-evm:~# cd /mnt/boot
root@keystone-evm:/mnt/boot# ls
uImage-k2hk-evm.dtb  uImage-keystone-evm.bin skern-keystone-evm.bin
```

The above files can be deleted and overwritten with new file. For example to replace the dtb file, do

```
root@keystone-evm:/mnt/boot# rm uImage-k2hk-evm.dtb
```

TFTP the uImage-k2hk-evm.dtb to this folder or using the DTC compiler on target DTS file may compiled and saved in this folder. Please note that the file name should match with default files in the boot volume.

Once done umount the folder as

```
root@keystone-evm:/# umount /mnt/boot
UBIFS: un-mount UBI device 0, volume 0 root@keystone-evm:/# reboot
```

SYS/BIOS

Placeholder for future

DSP Subsystem

Overview

The Multicore Software Development Kit (MCSDK) provides the core foundational building blocks that facilitate application software development on TI's high performance and multicore SoC. The foundational DSP components include:

- SYS/BIOS which is a light-weight real-time embedded operating system for TI devices
- Chip support libraries, drivers, and basic platform utilities
- Interprocessor communication for communication across cores and devices
- Basic networking stack and protocols
- Optimized application-specific and application non-specific algorithm libraries
- Debug and instrumentation
- Bootloaders and boot utilities
- Demonstrations and examples

The purpose of this *User's Guide* is to provide more detailed information regarding the software elements and infrastructure provided with MCSDK. MCSDK pulls together all the elements into demonstrable multicore applications and examples for supported EVMs. The objective being to demonstrate device, platform, and software capabilities and functionality as well as provide the user with instructive examples. The software provided is intended to be used as a reference when starting their development.

Note: In deciding the endianness of the DSP, all the features using little endian Linux configuration are only supported with Little endian DSP configuration in this release. Although some of the modules are tested standalone with Big endian DSP.

Note: It is expected the user has gone through the *EVM Quick Start Guide (TBD)* provided with their EVM and have booted the out-of-box demonstration application flashed on the device. It is also assumed the user has installed both CCS and the MCSDK.

API and LLD User Guides

API Reference Manuals and LLD User Guides are provided with the software. You can reference them from the Eclipse Help system in CCS or you can navigate to the components *doc* directory and view them there.

Tools Overview

The following documents provide information on the various development tools available to you.

Document	Description
CCS v5 Getting Started Guide (http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide)	How to get up and running with CCS v5
XDS560 Emulator Information (http://processors.wiki.ti.com/index.php/Xds_560)	Information on XDS560 emulator
XDS100 Emulator Information (http://processors.wiki.ti.com/index.php/XDS100)	Information on XDS100 emulator
TMS320C6000 Optimizing Compiler v 7.3 (http://focus.ti.com/lit/ug/spru187t/spru187t.pdf)	Everything you wanted to know about the compiler, assembler, library-build process and C++ name demangler.
TMS320C6000 Assembly Language Tools v 7.3 (http://focus.ti.com/lit/ug/spru186v/spru186v.pdf)	More in-depth information on the assembler, linker command files and other utilities.
Multi-core System Analyzer (http://processors.wiki.ti.com/index.php/MCSA)	How to use and integrate the system analyzer into your code base.
Eclipse Platform Wizard (http://rtsc.eclipse.org/docs-tip/Demo_of_the_RTSC_Platform_Wizard_in_CCSv4)	How to create a platform for RTSC. The demo uses CCSv4 but the platform screens are the same in CCSv5.
Runtime Object Viewer (http://rtsc.eclipse.org/docs-tip/Runtime_Object_Viewer)	How to use the Object Viewer for Eclipse Based Debugging.

Hardware - EVM Overview

The following documents provide information about the EVM.

Document	Description
Introducing the C66x Lite EVM Video (http://focus.ti.com/general/docs/video/Portal.tsp?entryid=0_55svdeqr&lang=en)	Short video on the C66x Lite Evaluation Module, the cost-efficient development tool from Texas Instruments that enables developers to quickly get started working on designs for the C66x multicore DSPs based on the KeyStone architecture.

Hardware - Processor Overview

The following documents provide information about the processor used on the EVM.

Document	Description
TCI6636K2H Data Manual (http://www.ti.com/lit/gpn/tci6636k2h)	Data manual for specific TI DSP

Related Software

This section provides a collection links to additional software elements that may be of interest.

Link	Description
C6x DSP Linux Project (http://www.linux-c6x.org/wiki/index.php/Main_Page)	Community site for C6x DSP Linux project
Telecom Libraries (http://focus.ti.com/docs/toolsw/folders/print/telecomlib.html)	TI software folder for information and download of Telecom Libraries (Voice, Fax, etc) for TI processors.
Medical Imaging Software Tool Kits (http://www.ti.com/tool/s2meddus)	TI software folder for information and download of medical imaging software tool kits for TI processors.

Platform Development Kit (PDK)

Chip Support Library (CSL)

The Chip Support Library constitutes a set of well-defined APIs that abstract low-level details of the underlying SoC device so that a user can configure, control (start/stop, etc.) and have read/write access to peripherals without having to worry about register bit-field details. The CSL services are implemented as distinct modules that correspond with the underlying SoC device modules themselves. By design, CSL APIs follow a consistent style, uniformly across Processor Instruction Set Architecture and are independent of the OS. This helps in improving portability of code written using the CSL.

CSL is realized as twin-layer – a basic register-layer and a more abstracted functional-layer. The lower register layer comprises of a very basic set of macros and type definitions. The upper functional layer comprises of "C" functions that provide an increased degree of abstraction, but intended to provide "directed" control of underlying hardware.

It is important to note that CSL does not manage data-movement over underlying h/w devices. Such functionality is considered a prerogative of a device-driver and serious effort is made to not blur the boundary between device-driver and CSL services in this regard.

CSL does not model the device state machine. However, should there exist a mandatory (hardware dictated) sequence (possibly atomically executed) of register reads/writes to setup the device in chosen "operating modes" as per the device data sheet, then CSL does indeed support services for such operations.

The CSL services are decomposed into modules, each following the twin-layer of abstraction described above. The APIs of each such module are completely orthogonal (one module's API does not internally call API of another module) and do not allocate memory dynamically from within. This is key to keeping CSL scalable to fit the specific usage scenarios and ease the effort to ROM a CSL based application.

The source code of the CSL is located under $(TI_PDK_INSTALL_DIR)\packages\ti\csl directory.

Note:

The CSL is build with LLD using same script, please refer the LLD build section for details

For KeyStone2 Devices CSL Package includes support for multiple devices. Software layer using CSL source would need to pass compile time define -DDEVICE_XXX. Refer ti\csl\cslr_device.h for list of devices/SOC's

Chip Support Library Summary	
Component Type	Library
Install Package	PDK
Install Directory	pdk_keystone2_<version>\packages\ti\csl
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little & Big
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\csl
Linker Sections	.vecs , .switch, .args, .cio
Section Preference	L2 Cache
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\csl
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Chip support library (http://processors.wiki.ti.com/index.php/CSL)
Downloads	Product Updates
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Low Level Drivers

The Low Level Drivers (LLDs) provide ineterfaces to the various peripherals on your SoC Device.

The source code for the LLDs is located under $(TI_PDK_INSTALL_DIR)\packages\ti\drv directory.

The following table shows PDK LLD vs. SoC Availability

Driver	Keystone2
CSL	X
QMSS	X
PKTDMA (CPPI)	X
PA	X
SA	X
SRIO	X
PCIe	X
Hyperlink	X
EDMA3	X
FFTC	X
TCP3d	X
BCP	X
RM	X
DFE	X
IQN2	X
TSIP	X
AIF2	X

Driver Library Summary	
Component Type	Library
Install Package	PDK
Install Directory	pdk_keystone2_<version>\packages\ti\drv
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little & Big
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\drv
Linker Sections	N/A
Section Preference	N/A
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\drv
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Chip support library (http://processors.wiki.ti.com/index.php/CSL)
Downloads	Product Updates
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Multicore Navigator

Multicore Navigator provides multicore-safe communication while reducing load on DSPs in order to improve overall system performance.

Packet DMA (CPPI)

The CPPI low level driver can be used to configure the CPPI block in CPDMA for the Packet Accelerator (PA). The LLD provides resource management for descriptors, receive/transmit channels and receive flows.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugr9c)
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\cppi\docs\CPPI_QMSS_LLD_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\cppi\docs\cppilldDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_CPPI_LLD.pdf

Queue Manager (QMSS)

The QMSS low level driver provides the interface to Queue Manager Subsystem hardware which is part of the Multicore Navigator functional unit for a keystone device. QMSS provides hardware assisted queue system and implements fundamental operations such as en-queue and de-queue, descriptor management, accumulator functionality and configuration of infrastructure DMA mode. The LLD provides APIs to get full entitlement of supported hardware functionality.

The LLD also includes accumulation and QoS (Quality of Service) firmware. QoS enables restriction of data rates in bytes per second or packets per second, weighted round robin queue selection, and selective descriptor dropping for oversubscribed queues. Accumulation The APIs are provided through the LLD. The API documentation for both QoS and Accumulator is available in the API Reference manual below, for all versions of MCSDK. The capabilities of the QoS firmware are documented in their design documents, which are present in MCSDK 3.1.4 and later. The capabilities of the accumulator are documented in the Hardware Peripheral User Guide.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugr9c)
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\CPPI_QMSS_LLD_SDS.pdf
QoS (Quality of Service) Design Document covering qos_sched_drop_sched, qos_sched, and qos_sched_wide.	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\firmware\qos_sched,qos_sched_drop_sched,qos_sched_wide.pdf
QoS (Quality of Service) Design Document covering qos (leaky bucket and SRIO TX context tracking).	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\firmware\qos.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\qmsslldDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_QMSS_LLD.pdf

Packet Library (PKTLIB)

Module expands underlying CPPI hardware descriptors for optimal usage at application layer.Functionality includes: - Zero Copy operations for Packet split/merge,Cloning - Headroom/Tail room addition through merge operations - Allocation of packet buffer and descriptors during startup time

Additional documentation can be found in:

Document	Location
Module Users Guide	$(TI_TRANSPORT_NET_LIB_INSTALL_DIR)\docs\TransportNetLib_UserGuide.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\runtime\pktlib\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\runtime\pktlib\docs\ReleaseNotes_pktlib.pdf

Network Co-processor (NETCP)

NETCP provides hardware accelerator functionality for processing Ethernet packets.

Security Accelerator (SA)

The SA also known as cp_ace (Adaptive Cryptographic Engine) is designed to provide packet security for IPsec, SRTP and 3GPP industry standards. The SA LLD provides APIs to abstract configuration and control between application and the SA. Similar to the PA LLD, it does not provide a transport layer. The Multicore Navigator is used to exchange control packets between the application and the SA firmware.

Note: Due to export control restrictions the SA driver is a separate download from the rest of the MCSDK.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugy6)
LLD Users Guide	$(TI_SA_LLD_<ver>_INSTALL_DIR)\sasetup\docs\UserGuide_SA_LLD.pdf
API Reference Manual	$(TI_SA_LLD_<ver>_INSTALL_DIR)\sasetup\packages\ti\drv\sa\docs\doxygen\sa_lld_docs.chm
Release	$(TI_SA_LLD_<ver>_INSTALL_DIR)\sasetup\packages\ti\drv\sa\docs\ReleaseNotes_SA_LLD.pdf

Notes	

Packet Accelerator (PA)

The PA LLD is used to configure the hardware PA and provides an abstraction layer between an application and the PA firmware. This does not include a transport layer. Commands and data are exchanged between the PA and an application via the Mutlicore Navigator.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugs4)
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\pa\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_PA_LLD.pdf

Network Adaptation Layer (NWAL)

Module provides higher level network adaptation layer, and abstracts NETCP access to all upper software layers in TransportNetLib package. Module uses API services from PA/SA/CPPI/QMSS/PKTLIB modules for control packets towards NetCP. Additionally module provides API for packet transmit and receive from NETCP

Additional documentation can be found in:

Document	Location
Module Users Guide covered in TransportNetLib User Guide	$(TI_TRANSPORT_NET_LIB_INSTALL_DIR)\docs\TransportNetLib_UserGuide.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\nwal\docs\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\nwal\docs\\ReleaseNotes_NWAL.pdf
User Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\nwal\docs\UserGuide_NWAL.pdf

I/O and Buses

Antenna Interface (AIF2)

The AIF2 low-level driver is meant to be used by applications which make use of AIF2, QMSS and CPPI IPs. The AIF2 low-level driver aims at generalizing the configuration of AIF2 for different modes (CPRI/OBSAI/Generic packet, WCDMA/LTE/Dual mode). The first goal of the AIF2 LLD is to provide programmers with a “functional layer” or an abstraction of the AIF2 configuration complexity. That means that within a short amount of configuration parameters / API calls, the end user can configure AIF2 for a specific high level scenario.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/ug/spruhl2a/spruhl2a.pdf)
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\aif2\docs\AIF2-c66xx_usersguide.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\aif2\docs\ AIF2-c66xx_apireferenceguide.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_AIF2_LLD.pdf

IQN2

IQN2 is intended to communicate with FFTC, RAC, TAC, PA/NetCP sub-systems without the need of constant supervision or control from application software. It is envisioned that application software would initially configure the interaction between IQN2 and these sub-systems, but that in steady state, no or very minimal CPU interventions are required for this data interchange.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/pdf/spruho6)
LLD SDS	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\iqn2\docs\IQN2_LLD_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\iqn2\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_IQN2_LLD.pdf

Digital Radio Front End (DFE)

DFE is a high performance wideband digital IF transmit and receive signal processing peripheral for small cell base station applications. It implements advanced algorithms for RF power amplifier linearization including crest factor reduction (CFR) and digital pre-distortion (DPD), and for correcting other receiver RF impairments like IQ imbalance, DC offset and distortion.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/pdf/spruhx8)
LLD SDS	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\dfe\docs\DFE_LLD_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\dfe\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_DFE_LLD.pdf

Serial RapidIO (SRIO)

The SRIO Low Level Driver provides a well defined standard interface which allows application to send and receive messages via the SRIO peripheral.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugw1)
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\srio\docs\SRIO_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\srio\docs\srioDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_SRIODriver.pdf

Peripheral Component Interconnect Express(PCIe)

The PCIe module supports dual operation mode: End Point (EP or Type0) or Root Complex (RC or Type1). This driver focuses on EP mode but it also provides access to some basic RC configuration/functionality. The PCIe subsystem has two address spaces. The first (Address Space 0) is dedicated for local application registers, local configuration accesses and remote configuration accesses. The second (Address Space 1) is dedicated for data transfer. This PCIe driver focuses on the registers for Address Space 0.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugs6a)
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\pcie\docs\pcieDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_PCIE_LLD.pdf

Hyperlink

The Hyperlink peripheral provides a high-speed, low-latency, and low-power point-to-point link between two Keystone (SoC) devices. The peripheral is also known as vUSR and MCM. Some chip-specific definitions in CSL and documentation may have references to the old names. The LLD provides a well defined standard interface which allows application to configure this peripheral.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugw8)
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\hyplnk\docs\hyplnkDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_HYPLNK_LLD.pdf

Co-processors

Bit-rate Coprocessor (BCP)

The BCP driver is divided into 2 layers: Low Level Driver APIs and High Level APIs. The Low Level Driver APIs provide BCP MMR access by exporting register read/write APIs and also provides some useful helper APIs in putting together BCP global and sub-module headers required by the hardware. The BCP Higher Layer provides APIs useful in submitting BCP requests and retrieving their results from the BCP engine.

Turbo Coprocessor Decoder (TCP3d)

The TCP3 decoder driver provides a well defined standard interface which allows application to send code blocks for decoding and receive hard decision and status via EDMA3 transfers.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugs0)
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\tcp3d\docs\TCP3D_DriverSDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\tcp3d\docs\TCP3D_DRV_APIIF.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\tcp3d\docs\ReleaseNotes_TCP3DDriver.pdf

FFT Accelerator Coprocessor(FFTC)

The FFTC driver is divided into 2 layers: Low Level Driver APIs and High Level APIs. The Low Level Driver APIs provide FFTC MMR access by exporting register read/write APIs and also provides some useful helper APIs in putting together FFTC control header, DFT size list etc. required by the hardware. The FFTC Higher Layer provides APIs useful in submitting FFT requests and retrieving their results from the FFTC engine without having to know all the details of the Multicore Navigator.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide (http://www.ti.com/lit/sprugs2b)
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\fftc\docs\FFTC_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\fftc\docs\fftcDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\fftc\docs\ReleaseNotes_FFTCDriver.pdf

Instrumentation

The instrumentation directory contains components that can be used for the purposes of DSP exception handling, logging, and debug.

Fault Management (FM) Library

The FM library defines a standard interface for a DSP to inform the Linux kernel of a fault and provide crash dump information. On a fault, a DSP configured to use the fault management library can shut down all data transfer IO and then write crash dump information to an ELF Note Section accessible to the Linux kernel. The fault-originating DSP core can use a FM API to send NMI pulses to other DSP cores to initiate exceptions via the DSP core NMI generation registers. Initiating exceptions on the remote DSP cores stops their processing to gain a potentially more complete picture of the system state when the originating-exception occurred. Lastly, the FM module provides an API that the exception handling routine can use to inform the Kernel the DSP has crashed due to an exception and that a coredump has been produced. This signal is sent to the ARM core via the ARM IPC interrupt generation register.

Note: Please note that sending NMI pulses to remote DSP cores to initiate remote exceptions is optional. Please do not initiate a call to this function from the exception hook function if crashing remote DSP cores is not needed.

Additional documentation can be found in:

Document	Location
FM Users Guide	FM Users Guide (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Developing_System_Mgmt#DSP_Fault_Management_.28FM.29)
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\fault_mgmt\docs\fault_mgmtlibDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\fault_mgmt\docs\ReleaseNotes_fault_mgmt.pdf

Watchdog Timer Module

The Watchdog Timer Module provides APIs for using the local DSP timers as watchdog timers. Each DSP has a local timer, typically DSP n's local timer is system timer n, that can be configured and used as a 64-bit watchdog timer for that DSP. The Watchdog Timer Module provides APIs for an application to configure and "feed" a DSPs watchdog timer as an exception failsafe. If the DSP were to crash or become corrupted the watchdog timer will timeout. The Watchdog Timer Module provides options for if the watchdog times out. The timer can be configured to perform a hard/soft reset, generate an NMI, and generate an exception via the BIOS Exception module.

If the Watchdog Timer module is configured to generate an exception through the BIOS Exception module it can be plugged into the Fault Management library to generate a core dump on exception.

Additional documentation can be found in:

Document	Location

Hardware Peripheral Users Guide	Timer64 User Guide (http://www.ti.com/lit/pdf/sprugv5)
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\wdtimer\docs\WatchdogTimerDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\wdtimer\docs\ReleaseNotes_WatchdogTimer.pdf

Trace framework (TF) Library

Trace framework is an infrastructure for sharing the information from a “single producer” to multi consumers (maximum 4) over shared memory, designed specifically for the needs of small cell.

Example use cases are:

1. UIA traces produced by a single producer, and consumed by other consumers. (Small cell uses this for instrumentation purposes).
2. Application produced proprietary information by one producer, consumed by multiple consumers in the system (Small cell uses this feature for OAM functionalities)

Additional documentation can be found in:

Document	Location
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\docs\ReleaseNotes_traceframework.pdf
User Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\docs\traceframework_ug.pdf

CSL MODIFICATIONS COMPARED TO KEYSTONE-I

- Multiple CSL macro define modifications corresponding to the register fields
- Several module specific MMR register fields are expanded which were arrays in overlay structures. For example in the modules COR, DNT, INT, etc.
- Data Structures changed - Bcp_DntHdrCfg, Bcp_SslHdr_WiMaxCfg, Bcp_IntHdrCfg.
- Following table captures register field changes at high level. Please refer the files for actual changes

Module	Removed	Added
SSL – HDR	tdd_output_pack field in WORD1 register	
TM – MMR		New STARVE registers (CDMA_DESC_STARVE_STATUS, CDMA_DESC_STARVE_CLEAR, CDMA_DESC_STARVE_SET, CDMA_DESC_STARVE_INTR_SEL)
TM – MMR		rx_teardown, tx_teardown, clkgate_disable fields in BCP_SOFT_RESET register
TM – MMR	rx_cdmahp_wr_arb_hpriority field in TM_CONTROL register	frc_payload_endian in TM_CONTROL register

LLD API MODIFICATIONS COMPARED TO KEYSTONE-1

Removed	Bcp_enableRxCdmaHpWriteArbPrio(), Bcp_disableRxCdmaHpWriteArbPrio(), Bcp_isRxCdmaHpWriteArbPrioEnabled()
Modified	Bcp_setTMControlReg(), Bcp_getTMControlReg()
Added	Bcp_doSoftResetRxTeardown(), Bcp_doSoftResetTxTeardown(), Bcp_doSoftResetClockGateDisable(), Bcp_enableForcePayloadEndian(), Bcp_disableForcePayloadEndian(), Bcp_isForcePayloadEndianEnabled(), Bcp_getCdmaDescStarveStatus(), Bcp_setCdmaDescStarveClear(), Bcp_setCdmaDescStarveSet(), Bcp_setCdmaDescStarveInterruptSelect()

Additional documentation can be found in:

Document	Location
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\bcp\docs\BCP_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\bcp\docs\bcpDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\bcp\docs\ReleaseNotes_BCPDriver.pdf

Keystone-II CSL/LLD API Migration

The following table shows API migration information for CSL/LLD:

KEYSTONE-II CSL/LLD API Migration								
		DSP PDK Deliverable		API Delta from KeyStone-I				
	Peripherals/IP Blocks	CSL	LLD	CSL-RL	CSL-Aux	LLD	Application Change Required	Notes for Migration/Compatibility
	Device Level (cslr_device.h)	x		YES			YES	Device specific support [MINOR - Register renaming required]
	BCP	x	x	YES	YES	YES	YES	Migration details are available in user guide [MAJOR]
	DDR3/EMIF4	x		YES	YES		YES	Change in IP [MAJOR]
	RAC	x		YES			YES	Change in IP [MAJOR]
	TAC	x		YES	YES		YES	Change in IP [MAJOR]
NetCP	EMAC (SGMII, CPSWITCH, MDIO)	x		YES	YES		YES	Support for 5-ports instead of 3-ports; CPSWITCH CSL-Aux layer APIs renamed from xxx_3GF_xxx of xxx_nGF_xxx [MINOR - Functiona renaming]
	SA	x	x	YES		YES	NO	Additional Crypto Engine; API Backward compatible
	PA	x	x	YES		NO	NO	Additional registers; API Backward compatible
	CPPI	x	x	NO		YES	NO	Support for 2 QM instances. No change required if application is using LLD provided cppi_device.c
	QMSS 1.5	x	x	NO	YES	YES	NO	Additional QMSS instance; API Backward compatible
	EDMA	x	x	NO		NO	NO	Additional EDMA instances are supported; API Backward compatible
	SRIO	x	x	NO		NO	NO	
	PCIe	x	x	NO		NO	NO	
	Hyperlink	x	x	NO		NO	NO	Additional registers; API Backward compatible
	FFTC	x	x	NO		NO	NO	
	TCP3D	x	x	NO		NO	NO	
	AIF 2.1	x		NO	YES		NO	Changes in CSL-FL for KeyStone-II
	AT	x		NO			NO	
	BCR	x		YES			NO	New IP
	CGEM	x		NO			NO	
	CHIP	x		NO	NO		NO	
	CP_BOOTCFG	x		YES	NO		NO	Register name change; API Backward compatible
	CP_INTC 0..2	x		NO	NO		NO	
	CP_MPU	x		NO	NO		NO	
	CP_TRACER	x		NO			NO	
	EMIF16	x		NO			NO	
	GPIO (2 banks)	x		NO			NO	
	I2C	x		NO			NO	
	IPC	x		NO			NO	
	MSMC	x		NO			NO	
	PLLCTRL	x		NO			NO	
	PSC	x		YES	NO		NO	Additional registers; API Backward compatible
	Semaphore	x		YES	NO		NO	Additional registers; API Backward compatible
	SPI	x		NO			NO	
	TIMER64P	x		YES	NO		NO	Additional registers; API Backward compatible
	TSIP			NO			NO	
	UART	x		NO			NO	
	USB3 SS	x		YES			NO	New IP
	USIM	x		YES			NO	New IP
	VCP	x		NO	NO		NO	
	XMC	x		NO	NO		NO	
	XGE (10 Gig 5 Port Switch)	x		YES			NO	New IP

Platform Library

The platform library defines a standard interface for platform utilities and functionality and provides sample implementations for the EVM platform. These include things such as reading and writing to EEPROM, FLASH, UART, etc. Platform library supports three libraries

1. debug library (ti.platform.evm###.ae66) - located under \platform_lib\lib\debug, needed only when a debug is needed on the platform library since the source is compiled with full source debugging.
2. release library (ti.platform.evm###.ae66) - located under \platform_lib\lib\release, should be used normally for the best performance of the cycles since the code is compiled with the full optimization.

Platform Library Summary	
Component Type	Library
Install Package	PDK for Keystone II
Install	pdk_keystone2_<version>\packages\ti\platform\evm###\platform_lib

Directory	
Project Type	CCS (http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide)
Endian Support	Little
Library Name	Select for the TCI66##### EVM ti.platform.evm###.ae66 (little)
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\platform\evm###\platform_lib\lib\debug - for debug version $(TI_PDK_INSTALL_DIR)\packages\ti\platform\evm###\platform_lib\lib\release - for release version
Linker Sections	platform_lib
Section Preference	none
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\platform platform.h defines the interface
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Texas Instruments Embedded Processors Wiki (http://processors.wiki.ti.com/)
Downloads	Product Updates
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Transport

The Transports are intermediate drivers that sit between either the NDK and interface them to the appropriate EVM peripherals. The Transport supported by MCSDK are:

- NDK transport - Network Interface Management Unit (NIMU) Driver

More information on these can be found in the NDK or IPC sections of this guide.

EDMA3 Low Level Driver

EDMA3 Low Level Driver is targeted to users (device drivers and applications) for submitting and synchronizing EDMA3-based DMA transfers.

EDMA3 is a peripheral that supports data transfers between two memory mapped devices. It supports EDMA as well as QDMA channels for data transfer. This peripheral IP is re-used in different SoCs with only a few configuration changes like number of DMA and QDMA channels supported, number of PARAM sets available, number of event queues and transfer controllers etc. The EDMA3 peripheral is used by other peripherals for their DMA needs thus the EDMA3 Driver needs to cater to the requirements of device drivers of these peripherals as well as other application software that may need to use DMA services.

The EDMA3 LLD consists of an EDMA3 Driver and EDMA3 Resource Manager. The **EDMA3 Driver** provides functionality that allows device drivers and applications for submitting and synchronizing with EDMA3 based DMA transfers. In order to simplify the usage, this component internally uses the services of the **EDMA3 Resource Manager** and provides one consistent interface for applications or device drivers.

EDMA3 Driver Summary	
Component Type	Library
Install Package	EDMA3 Low level drivers
Install Directory	<root_install_dir>/edma3_lld_xx_xx_xx_xx
Project Type	N/A
Endian Support	Little and Big
Library Name	edma3_lld_drv.ae66 (little endian) and edma3_lld_drv.ae66e (big endian)
Linker Path	N/A
Linker Sections	N/A
Section Preference	N/A
Include Paths	N/A
Reference Guides	See docs under install directory
Support	Technical Support
Additional Resources	Programming the EDMA3 using the Low-Level Driver (LLD) (http://processors.wiki.ti.com/index.php/Programming_the_EDMA3_using_the_Low-Level_Driver_%28LLD%29)
Downloads	Product Updates
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

SYS/BIOS

SYS/BIOS is a scalable real-time kernel. It is designed to be used by applications that require real-time scheduling and synchronization or realtime instrumentation. SYS/BIOS provides preemptive multi-threading, hardware abstraction, real-time analysis, and configuration tools. SYS/BIOS is designed to minimize memory and CPU requirements on the target.

SYS/BIOS Summary	
Component Type	Libraries
Install Package	SYS/BIOS
Install Directory	bios_6_<version>\
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little and Big
Library Name	The appropriate libraries are selected for your device and platform as set in the RTSC build properties for your project and based on the use module statements in your configuration.
Linker Path	The appropriate path is selected to the libraries for your device and platform as set in the RTSC build properties for your project.
Linker Sections	N/A
Section Preference	N/A
Include Paths	BIOS_CG_ROOT is set automatically by CCS based on the version of BIOS you have checked to build with. ${BIOS_CG_ROOT}\packages\ti\bios\include
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	SYS/BIOS Online Training (http://processors.wiki.ti.com/index.php/SYS/BIOS_Online_Training) SYS/BIOS 1.5-DAY Workshop (http://processors.wiki.ti.com/index.php/SYS/BIOS_1.5-DAY_Workshop) Eclipse RTSC Home (http://www.eclipse.org/rtsc/)
Downloads	SYS/BIOS Downloads (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/index.html)
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Inter-Processor Communication (IPC)

Inter-Processor Communication (IPC) provides communication between processors in a multi-processor environment.

IPC provides messaging APIs for Linux-to-BIOS and BIOS-to-BIOS communication.

IPC Summary	
Component Type	Libraries
Install Package	IPC
Install Directory	ipc_<version>
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little and Big
Library Name	The appropriate libraries are selected based on your application's configuration.
Linker Path	The appropriate paths are added in the config-generated linker command script, based on your application's configuration.
Linker Sections	N/A
Section Preference	N/A
Include Paths	IPC_INSTALL_DIR/packages
Reference Guides	IPC Users Guide API Reference (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/latest/docs/doxygen/html/index.html) Config Reference (http://software-dl.ti.com/dsps/dsps_public_sw/sdo

	_sb/targetcontent/ipc/latest/docs/cdoc/index.html)
Support	Technical Support
Additional Resources	Eclipse RTSC Home (http://www.eclipse.org/rtsc/)
Downloads	IPC Downloads (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/index.html)
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

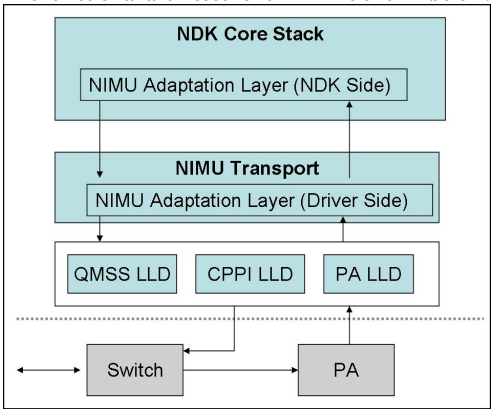
Network Development Kit (NDK)

The NDK is a platform for development and demonstration of network enabled applications on DSP devices and includes demonstration software showcasing DSP capabilities across a range of network enabled applications. The NDK serves as a rapid prototype platform for the development of network and packet processing applications, or to add network connectivity to existing DSP applications for communications, configuration, and control. Using the components provided in the NDK, developers can quickly move from development concepts to working implementations attached to the network.

The NDK provides an IPv6 and IPv4 compliant TCP/IP stack working with the SYS/BIOS real-time operating system. Its primary focus is on providing the core Layer 3 and Layer 4 stack services along with additional higher-level network applications such as HTTP server and DHCP.

The NDK itself does not include any platform or device specific software. The NDK interfaces through well-defined interfaces to the PDK and platform software elements needed for operation.

The functional architecure for NDK is shown below.



Network Development Kit Summary	
Component Type	Libraries
Install Package	NDK
Install Directory	ndk_<version>\
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little and Big
Library Name	binsrc.lib or binsrce.lib and cgi.lib or cgie.lib and console.lib or consolee.lib and hdlc.lib or hdlce.lib and miniPrintf.lib or miniPrintfe.lib and netctrl.lib or netctrle.lib and nettool.lib or nettoole.lib and os.lib or ose.lib and servers.lib or serverse.lib and stack.lib or stacke.lib
Linker Path	$(NDK_INSTALL_DIR)\packages\ti\ndk\lib\<arch>
Linker Sections	.far:NDK_OBJMEM, .far:NDK_PACKETMEM
Section Preference	L2 Cache
Include Paths	NDK_INSTALL_DIR is set automatically by CCS based on the version of NDK you have checked to build with. ${NDK_INSTALL_DIR}\packages\ti\ndk\inc ${NDK_INSTALL_DIR}\packages\ti\ndk\inc\tools
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	The NDK unit test examples are available in $(TI_MCSDK_INSTALL_DIR)\packages\ti\platform\nimu\test\evm####
Extended Support	Eclipse RTSC Home (http://www.eclipse.org/rtsc/) NDK User's Guide (http://www-s.ti.com/sc/techlit/spru523.pdf) NDK Programmer's Reference Guide (http://www-s.ti.com/sc/techlit/spru524.pdf) NDK Support Package Ethernet Driver Design Guide (http://www-s.ti.com/sc/techlit/sprufp2.pdf) NDK_FAQ (http://processors.wiki.ti.com/index.php/Network_Developers_Kit_FAQ) Rebuilding NDK Core (http://processors.wiki.ti.com/index.php/Rebuilding_the_NDK_Core)
Downloads	NDK Downloads (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ndk/index.html)
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Network Interface Management Unit (NIMU) Driver

NIMU sits between NDK common software and the TCI66##### platform library and provides a common interface for NDK to communicate with. This package have NDK unit test examples for all supported platforms.

Note: This module is only intended to be used with NDK. As such, users should not tie up to its API directly.

The functional architecture for NIMU (taking TCI66##### platform as an example) is shown below.

NIMU Summary	
Component Type	Library
Install Package	PDK_INSTALL_DIR
Install Directory	mcsdk_<version>\packages\ti\transport\ndk\nimu
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little
Library Name	ti.transport.ndk.nimu.ae66 (little)
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\transport\ndk\nimu\lib\debug for debug version $(TI_PDK_INSTALL_DIR)\packages\ti\transport\ndk\nimu\lib\release for release version
Linker Sections	nimu_eth_ll2
Section Preference	L2SRAM
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\transport\ndk\nimu\include
Reference Guides	None
Support	Technical Support
Additional Resources	The NDK unit test examples are available in $(TI_MCSDK_INSTALL_DIR)\examples\ndk\evm###
Downloads	http://focus.ti.com/docs/toolsw/folders/print/bioslinuxmcsdk.html
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Algorithm Libraries

TI provides several algorithm libraries, each specific to a particular arena. Each library provides a collection of C-callable low-level functions (kernels), each tailored for optimal performance on a specific TI processing device (or devices). The libraries are typically used in computationally intensive real-time applications where execution speed is a critical factor. Their use generally accelerates execution speeds well beyond that achieved by equivalent code written in standard ANSI C. Additionally, use of these libraries can significantly reduce application development time. Source code is provided in all cases to facilitate kernel modification when needed.

See c6x Software Library mediawiki (http://processors.wiki.ti.com/index.php/Software_libraries) for a comprehensive overview of the various software libraries available for TI's c6x family of processors.

DSP Library (DSPLIB)

DSPLIB is an optimized DSP Function Library and includes many C-callable, optimized, general-purpose signal-processing routines including:

- Adaptive Filtering
- Correlation
- Fast Fourier Transform
- Filtering and convolution
- Matrix

DSPLIB Summary	
Component Type	Library
Install Package	DSPLIB
Install Directory	dsplib_c66x_<version>\
Project Type	CCS (http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide)
Endian Support	Big and Little
Library Name	dsplib.a66 (COFF, little-endian) dsplib.a66e (COFF, big-endian) dsplib.ae66 (ELF, little-endian) dsplib.ae66e (ELF, big-endian)
Linker Path	<root_install_dir>\lib\
Linker	N/A

Sections	
Section Preference	N/A
Include Paths	<root_install_dir>\inc\ <root_install_dir>\packages\
Reference Guides	See docs under Install Directory
Support	BIOS E2e Forum (http://e2e.ti.com/support/embedded/f/355.aspx)
Additional Resources	c6x Software Library mediawiki (http://processors.wiki.ti.com/index.php/Software_libraries)
Downloads	DSPLIB Downloads (http://software-dl.ti.com/sdoemb/sdoemb_public_sw/dsplib/latest/index_FDS.html)
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Image Processing Library (IMGLIB)

IMGLIB is an optimized image/video processing library with kernels in the following functional categories:

- Compression & Decompression
- Image Analysis
- Image Filtering and Conversion

IMGLIB Summary	
Component Type	Library
Install Package	IMGLIB
Install Directory	imglib_c66x_<version>\
Project Type	CCS (http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide)
Endian Support	Little
Library Name	imglib.ae66 (ELF, little-endian)
Linker Path	<root_install_dir>\lib\
Linker Sections	N/A
Section Preference	N/A
Include Paths	<root_install_dir>\inc\ <root_install_dir>\packages\
Reference Guides	See docs under Install Directory
Support	BIOS E2e Forum (http://e2e.ti.com/support/embedded/f/355.aspx)
Additional Resources	c6x Software Library mediawiki (http://processors.wiki.ti.com/index.php/Software_libraries)
Downloads	IMGLIB Downloads (http://software-dl.ti.com/sdoemb/sdoemb_public_sw/imglib/latest/index_FDS.html)
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

Floating Point Math Library (MATHLIB)

MATHLIB contains optimized versions of most commonly used floating point math routines contained in the RTS library. Kernels are offered in two variations:

- Double-precision floating point
- Single-precision floating point

MATHLIB Summary	
Component Type	Library
Install Package	MATHLIB
Install Directory	mathlib_c66x_<version>\
Project Type	CCS (http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide)
Endian Support	Big and Little
Library Name	mathlib.a66 (COFF, little-endian) mathlib.a66e (COFF, big-endian) mathlib.ae66 (ELF, little-endian) mathlib.ae66e (ELF, big-endian)
Linker Path	<root_install_dir>\lib\
Linker	N/A

Sections	
Section Preference	N/A
Include Paths	<root_install_dir>\inc\ <root_install_dir>\packages\
Reference Guides	See docs under Install Directory
Support	BIOS E2e Forum (http://e2e.ti.com/support/embedded/f/355.aspx)
Additional Resources	c6x Software Library mediawiki (http://processors.wiki.ti.com/index.php/Software_libraries)
Downloads	MATHLIB Downloads (http://focus.ti.com/docs/toolsw/folders/print/mathlib.html)
License	BSD (http://www.opensource.org/licenses/bsd-license.php)

OMP

Resource Manager

The Resource Manager is detailed here: RM User Guide (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Developing_System_Mgmt#Resource_Manager)

DSP Management

Boot Utilities

Overview

MCSDK includes a Tools Package which provides boot utilities for use with the TI EVMs and are intended to serve as example/reference for customers.

The MCSDK tools package is located in the C:\ti\mcsdk_bios_#_##_##_##\tools directory and includes:

- **Writer Utilities**: Utilities to program an application image to flash or EEPROM.
- **Program EVM**: Utilities to program default application images to flash.
- **Other Utilities**: Utilities to do file format conversion that are required by the boot examples.

Flash and Flash Utilities

The following boot utilities for loading code into the EEPROM, NOR and NAND are provided as part of the Tools Package with the MCSDK. All source code is provided along with documentation so that customers can port to other environments as necessary or to make modifications and enhancements.

- **EEPROM Writer**: Utility for writing to the EEPROM. This utility executes on the EVM using CCS and JTAG and it is located under C:\ti\mcsdk_bios_#_##_##_##\tools\writer\eeprom\evm###\bin directory.
- **NOR Writer**: Utility for writing to the NOR flash. This utility executes on the EVM using CCS and JTAG and it is located under C:\ti\mcsdk_bios_#_##_##_##\tools\writer\nor\evm###\bin directory.
- **NAND Writer**: Utility for writing to the NAND flash. This utility executes on the EVM using CCS and JTAG and it is located under C:\ti\mcsdk_bios_#_##_##_##\tools\writer\nand\evm###\bin directory.

Useful Tip

The program_evm utility provides the ability to format the NAND (i.e., permanently erase the entire NAND device). Please refer to program_evm_userguide.pdf (located in the mcsdk_bios_#_##_##_##\tools\program_evm\ directory) for more information.

c6x DSP Linux Community

(point to community page) ß Not required since not applicable to KeyStone II as of yet, but just a test of scalability of the document.

DSP Acceleration,Profiling utilities and Libraries

The K2H file system comes bundled with a bunch of applications below

DSP Acceleration

The following components helps the applications to offload their computation to the DSPs. These runtime components are detailed below

OpenCL

OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, DSPs and other processors. OpenCL is used to dispatch tasks from A15 to DSP cores . More details could be found here (http://downloads.ti.com/mctools/esd/docs/opencl/index.html)

OpenMP

OpenMP is the de facto industry standard for shared memory parallel programming. OpenMP is used to achieve parallelism across DSP cores. More details could be found here (http://processors.wiki.ti.com/index.php/MCSDK_HPC_3.x_OpenMP)

DSP Debug and Profiling Utilities

The following Debug & profiling utilities are included for K2H.

DSPTOP

DSPTOP is similar to the Linux top utility and provides visibility to usage data for TI multicore DSP-ARM SoC devices. More details could be found here (http://processors.wiki.ti.com/index.php/Dsptop)

C66x GDB

C66x GDB is an implementation of a GDB Server debugging stub for the Texas Instruments C66x DSP. It allows developers to utilize the standard features of GDB (GNU Debugger) to gain visibility to and debug the C66x DSP cores in the heterogeneous DSP + ARM KeyStone II system-on-chips. More details could be found here (http://processors.wiki.ti.com/index.php/Hosted_C66x_GDB)

Libraries

The K2H file system comes with libraries optimized for TI SOC and could be used for application development.

FFTW

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data. More details on the TI implementation of FFTW could be found here (http://processors.wiki.ti.com/index.php/MCSDK_HPC_3.x_FFTW_Library)

Linear Algebra

The TI Linear Algebra library (LINALG) is an optimized library for performing dense linear algebra computations. More details could be found here (http://processors.wiki.ti.com/index.php/MCSDK_HPC_3.x_Linear_Algebra_Library)

Machine Learning Libraries

The TI Machine Learning (TIML) Library is a machine learning library currently focused on deep learning networks in general and convolutional neural networks in particular. The machine learning library runs on the ARM and relies on lower level DSP libraries for acceleration. More details could be found here (http://processors.wiki.ti.com/index.php/MCSDK_HPC_3.x_Machine_Learning_Library)

Graph Libraries

The TI Graph Library (TIGL) is a template graph library that supports the Pregel API. More details could be found here (http://processors.wiki.ti.com/index.php/MCSDK_HPC_3.x_Graph_Library)

NOTE: Please note that the above are available only for K2H

NOTE: For MCSDK-HPC 3.0.1.x users ,please refer to this page Migration Guide (http://processors.wiki.ti.com/index.php?title=MCSDK_HPC_MigrationGuide_to_MCSDK3.1.4)

Tools

Multicore System Analyzer (MCSA)

Multicore System Analyzer (MCSA) is a suite of tools that provide real-time visibility into the performance and behavior of your code, and allow you to analyze information that is collected from software and hardware instrumentation in a number of different ways.

Advanced Tooling Features:

- Real-time event monitoring
- Multicore event correlation
- Correlation of software events, hardware events and CPU trace
- Real-time profiling and benchmarking
- Real-time debugging

Two key components of System Analyzer are:

- DVT: Various features of Data Analysis and Visualization Technology (DVT) provide the user interface for System Analyzer within Code Composer Studio (CCS)
- UIA: The Unified Instrumentation Architecture (UIA) target package defines APIs and transports that allow embedded

MCSA Summary	
Component Type	Libraries
Install Package	UIA + DVT
Install Directory	ccsv5/uia_<version>, ccsv5/eclipse, ccsv5/ccs_base_5.0.0.*/dvt\
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little
Library Name	The appropriate libraries are selected for your device and platform as set in the RTSC build properties for your project and based on the use module statements in your configuration.
Linker Path	The appropriate path is selected to the libraries for your device and platform as set in the RTSC build properties for your project.

software to log instrumentation data for use within CCS

Linker Sections	N/A
Section Preference	N/A
Include Paths	N/A
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Multicore System Analyzer (http://processors.wiki.ti.com/index.php/Multicore_System_Analyzer)
Downloads	Installed as a part of MCSDK installation
UIA License	BSD (http://www.opensource.org/licenses/bsd-license.php)
DVT License	TI Technology and Software Publicly Available (TSPA). See DVT Manifest in the install directory.

Eclipse RTSC Tools (XDC)

RTSC is a C-based programming model for developing, delivering, and deploying Real-Time Software Components targeted for embedded platforms. The XDCtools product includes tooling and runtime elements for component-based programming using RTSC.

XDC Summary	
Component Type	Tools
Install Package	XDC
Install Directory	xdctools_<version>\
Project Type	Eclipse RTSC (http://www.eclipse.org/rtsc/)
Endian Support	Little and Big
Library Name	The appropriate libraries are selected for your device and platform as set in the RTSC build properties for your project and based on the use module statements in your configuration.
Linker Path	The appropriate path is selected to the libraries for your device and platform as set in the RTSC build properties for your project.
Linker Sections	systemHeap
Section Preference	none
Include Paths	N/A
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Eclipse RTSC Home (http://www.eclipse.org/rtsc/) Users Guide and Reference Manual (http://rtsc.eclipse.org/docs-tip/Main_Page)
Downloads	N/A
License	See XDC Manifest in the install directory

Demonstrations

The MCSDK consist of demonstration software to illustrate device and software capabilities, benchmarks, and usage.

See the Getting Started Guide's demonstration section (http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Getting_Started#Running_Out_of_Box_Demonstrations) for instructions to run the pre-built demonstrations in the filesystem.

Utility Application

http://processors.wiki.ti.com/index.php/MCSDK_Utility_App_Demonstration_Guide

Image Processing

http://processors.wiki.ti.com/index.php/MCSDK_Image_Processing_Demonstration_Guide

Inter-Processor Communication

http://processors.wiki.ti.com/index.php/MCSDK_Inter_Processor_Communication_Demonstration_Guide

Running examples of OpenCL,OpenMP and Libraries

The MCSDK filesystem for K2H comes bundled with examples for OpenCL,OpenMP and libraries which are typically installed in /usr/share/ti/examples/<component> directory. The following steps are needed to run them the K2H EVM

After mounting the K2H file system,

Add user to 'keystone' group

A non-root user needs to be a part of the 'keystone' group for accessing peripherals such as qmss,dsp etc. The 'keystone' group aleady exists in the K2H file system and the non-root user needs to be added to that group using the below command. For example, if the non-root user is "evm-user"

```
su
usermod -a -G keystone evm-user
```

The EVM needs to be rebooted after this step.

Compile the examples

The examples for various components are stored under /usr/share/ti/examples/<component-name>. For example, opencl demos are kept at /usr/share/ti/examples/opencl To compile a component's demos, do the following

Copy the example to a directory

For example, in case of opencl

```
cp -r /usr/share/ti/examples/opencl  ~/
```

Run the make command

Run the make command at the top of the directory

```
cd ~/opencl
make
```

This will build all the examples bundled with opencl. To compile a specific example, you may go in to that sub-directory and issue a make command For example

```
cd ~/opencl/null
make
```

The above builds the 'null' example alone.

Run the examples

The examples built in the above step binaries reside in their respective directories. To run any specific example manually, for example the 'null' example,

```
cd ~/opencl/null/
./null
```

The above test's output is shown below

```
The OpenCL runtime will lazily load the device program upon the
first enqueue of a kernel from the program, so the elapsed time
overall from the first enqueue will be longer to account for the
loading of the kernel.  Also, subsequent enqueue's of a kernel will
also potentially benefit from a warm cache on the device.
Elapsed (with Load): 561 usecs
Elapsed (w/o  Load): 69 usecs
Elapsed (w/o  Load): 129 usecs
Elapsed (w/o  Load): 149 usecs
Elapsed (w/o  Load): 145 usecs
Elapsed (w/o  Load): 142 usecs
Done!
```

Running Automated Tests

The K2H filesystem comes bundled with an automated test suite which lets us test all the examples (OpenCL,OpenMP and Libraries) in one step. This automated test suite builds, runs all the examples and generate a test report as well. This test suite also provides logs for individual tests as well.

To run the automated test do the following,

Copy the autotest files

Copy the /usr/share/ti/keystone-hpc/autotest folder to the current directory

```
> cp -r /usr/share/ti/keystone-hpc/autotest .
```

Run the automated test

```
> cd autotest
> ./test_release.sh --run-on-hosts <hostname>
```

For example

```
> ./test_release.sh --run-on-hosts k2hk-evm
```

Part of the above command involves copying the demo examples to the current directory, recompiling them. So, if you would like to skip the portion of recompiling the examples (for example, to just run the test on the binaries produced in an earlier run), you could do it with the 'skip-rebuild' option appended to the script. So, the test command would look like

```
> ./test_release.sh --run-on-hosts skip-rebuild <hostname>
```

For example

```
> ./test_release.sh --run-on-hosts skip-rebuild k2hk-evm
```

Please note that skip-rebuild should be used ONLY if you had successfully run the automated tests at least once before from the same location (with no skip-rebuild option).

Verify the test results

The test takes about 20 minutes or so. The results of the test against each test ,are printed on screen like below,

```
> "opencl:platforms" ... [PASS]
> "opencl:edmamgr" ... [PASS]
> "opencl:ccode" ... [PASS]
> "opencl:dsplib_fft" ... [PASS]
> "opencl:simple" ... [PASS]
> "opencl:null" ... [PASS]
>  Complete
```

NOTE: The following failures are expected.

```
> "opencl:ooo" ...                                                    [FAIL]
> "opencl:mandelbrot" ...                                             [FAIL]
> "opencl:mandelbrot_native" ...                                      [FAIL]
```

The logs of test execution are kept under seperate folder created every run, based on the timestamp. The folder name is printed at the beginning of the test, For example,

```
> CONFIG: HOSTS_FILE = /home/evmuser/autotest/hosts
> CONFIG: TEST_USER = evmuser
> CONFIG: EXAMPLES_DIR = /home/evmuser/autotest/mcsdk-hpc-examples
> CONFIG: NODE_KNOWN_HOSTS = /home/evmuser/.ssh/known_hosts
> mkdir: created directory /home/evmuser/autotest/20150202211040
> mkdir: created directory /home/evmuser/autotest/20150202211040/test_logs ---> This is the directory where the logs are present for each test case.
> mkdir: created directory /home/evmuser/autotest/20150202211040/logs
```

In the above example, the test logs for a particular run is kept at 20150202211040/test_logs directory

How To

How to run Linux kernel from a different physical address than default (0x80000000) ?

A sample patch to change DDR base address to 0xA0000000 instead of the default address 0x80000000 is provided below. User is responsible for maintaining this patch at their code base and the same will not be included in the MCSDK.

Linux code patch

```
--- ref/keystone-0430a//linux-keystone/arch/arm/mach-keystone/Makefile.boot     2013-04-30 11:19:07.000000000 -0700
+++ linux-keystone/arch/arm/mach-keystone/Makefile.boot 2013-05-09 13:47:29.829257106 -0700
@@ -1 +1 @@
-zreladdr-y     := 0x80008000
+zreladdr-y     := 0xA0008000
--- ref/keystone-0430a//linux-keystone/arch/arm/mach-keystone/include/mach/memory.h     2013-04-30 11:19:07.000000000 -0700
+++ linux-keystone/arch/arm/mach-keystone/include/mach/memory.h 2013-05-09 12:05:10.561756528 -0700
@@ -16,6 +16,7 @@
 #ifndef __ASM_MACH_MEMORY_H
 #define __ASM_MACH_MEMORY_H

+
 #define MAX_PHYSMEM_BITS       36
 #define SECTION_SIZE_BITS      34

@@ -32,10 +33,17 @@

 #ifndef __ASSEMBLY__

+static inline phys_addr_t __virt_to_phys(unsigned long x);
+
 static inline phys_addr_t __virt_to_idmap(unsigned long x)
```

```
 {
+#ifdef ORIGINAL_PRE_TBESEMER
        return (phys_addr_t)(x) - CONFIG_PAGE_OFFSET +
                KEYSTONE_LOW_PHYS_START;
+#else
+       return __virt_to_phys(x) - KEYSTONE_HIGH_PHYS_START +
+               KEYSTONE_LOW_PHYS_START;
+#endif
 }

 #define virt_to_idmap(x)       __virt_to_idmap((unsigned long)(x))
```

Device tree source patch

```
--- ref/keystone-0430a//linux-keystone/arch/arm/boot/dts/k2hk-evm.dts   2013-04-30 11:19:07.000000000 -0700
+++ linux-keystone/arch/arm/boot/dts/k2hk-evm.dts       2013-05-10 09:56:40.418347675 -0700
@@ -17,7 +17,7 @@
        };

        memory {
-               reg = <0x00000000 0x80000000 0x00000000 0x20000000>;
+               reg = <0x00000000 0xA0000000 0x00000000 0x20000000>;
        };

        droppolicies: default-drop-policies {
```

u-boot code patch

```
--- ref/keystone-0430a//u-boot-keystone/include/configs/tci6638_evm.h   2013-04-30 10:29:03.000000000 -0700
+++ u-boot-keystone/include/configs/tci6638_evm.h       2013-05-09 13:46:25.277274692 -0700
@@ -47,8 +47,8 @@

 /* Memory Configuration */
 #define CONFIG_NR_DRAM_BANKS           1
-#define CONFIG_SYS_SDRAM_BASE          0x80000000
-#define CONFIG_MAX_RAM_BANK_SIZE       (2 << 30)       /* 2GB */
+#define CONFIG_SYS_SDRAM_BASE          0xA0000000
+#define CONFIG_MAX_RAM_BANK_SIZE       ((1024 * 1024) * 1024)
 #define CONFIG_STACKSIZE               (512 << 10)     /* 512 KiB */
 #define CONFIG_SYS_MALLOC_LEN          (512 << 10)     /* 512 KiB */
 #define CONFIG_SYS_MEMTEST_START       CONFIG_SYS_SDRAM_BASE
@@ -194,7 +194,7 @@
 #define CONFIG_CMD_SF

 /* U-Boot general configuration */
-#define CONFIG_SYS_PROMPT              "TCI6638 EVM # "
+#define CONFIG_SYS_PROMPT              "TCI6638 EVM (teb) # "
 #define CONFIG_SYS_CBSIZE              1024
 #define CONFIG_SYS_PBSIZE              2048
 #define CONFIG_SYS_MAXARGS             16
@@ -265,6 +265,6 @@
 #define CONFIG_SETUP_MEMORY_TAGS
 #define CONFIG_SYS_BARGSIZE            1024
 #define CONFIG_SYS_LOAD_ADDR           (CONFIG_SYS_SDRAM_BASE + 0x08000000)
-#define LINUX_BOOT_PARAM_ADDR          (CONFIG_SYS_SDRAM_BASE + 0x100)
+#define LINUX_BOOT_PARAM_ADDR          (CONFIG_SYS_SDRAM_BASE + 0x00000100)

 #endif /* __CONFIG_H */
```

Also when using the patch, make sure to update the following env variables in u-boot

```
setenv addr_fdt 0xA7000000
setenv addr_kern 0xA8000000
```

How to boot with a combined kernel and initramfs image?

This section is borrowed from [[13] (http://processors.wiki.ti.com/index.php/Initrd)]

To use initramfs as part of a combined kernel/initramfs image, a cpio archive is embedded directly into the kernel. I.e. you don't create an additional ramfs image. Instead, the initial file system is directly incorporated into the kernel. With this, the kernel size increases by the file system size. It's like you embed above ramfs directly into the kernel. You simply have to fill a directory on your host with the target filesystem you like and then pass the path to this directory to the kernel build process.

Create target file system

```
host > mkdir target_fs
host > ... copy stuff you want to have in initramfs to target_fs... do following command using arago-console-image.tar.gz provided in the release
host > cd target_fs
host > sudo tar -xvzf <path of arago-console-image.tar.gz>
host > cd ..
host > sudo chmod a+rwx target_fs -R
```

Now cd to linux kernel root directory.

Configure kernel to embed initramfs

Kernel options

Now give the kernel the path to the target file system you like to embed. Edit the defconfig for the platform and update the CONFIG_INITRAMFS_SOURCE variable and save:-

```
#
# General setup
#
...
CONFIG_BLK_DEV_INITRD=y
CONFIG_INITRAMFS_SOURCE="<path_to>/target_fs>"
...
```

```
#
# UBI - Unsorted block images
#
...
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=1
CONFIG_BLK_DEV_RAM_SIZE=8192
...
```

Then, if you compile the kernel, e.g. by make uImage, the cpio archive is generated and embedded into the kernel:

```
...
CHK     include/linux/compile.h
GEN     usr/initramfs_data.cpio.gz
AS      usr/initramfs_data.o
LD      usr/built-in.o
...
```

Setup u-boot environment and boot combined image

At the u-boot console, do the following:-

For boot mode = ramfs do the following:-

Copy the kernel image (build using procedure above), uImage-combined, to the tftp server root directory.

```
env default -f -a
setenv serverip <tftp server IP>
setenv args_ramfs 'setenv bootargs ${bootargs} earlyprintk rdinit=/sbin/init'
setenv name_kern uImage-combined
setenv tftp_root <tftp server root directory>
setenv init_ramfs 'run args_all args_ramfs'
saveenv
```

At this time, issue the boot command and Linux boots up with embedded initramfs image.

How to change Tetris and Core PLL speed?

The prerequisite is to follow hardware setup guide to update EVM with the latest BMC and UCD firmware version.

1. The PLL clock frequency is now programmed based on EFUSE register settings for K2HK Rev. 3.x EVM and K2EL Rev. 1.0.2.x. The getclk command in u-boot can be used to know the actual frequency used. The default PLL speeds of the Core(DSP) and Tetris(ARM) are

Platform	Core	Tetris
K2H	800MHz	1200MHz
K2K	800MHz	1200MHz
K2E	1250MHz	1250MHz
K2L	1250MHz	1250MHz

2. To change the Tetris frequency, the EVM's reference clock is needed for the calculation. Each platform may have different main reference clock frequency, please refer to the sys_clk defined in board_k2x.c for the refclkmain frequency.

If Tetris PLL speed needs to be changed on K2HK, at Uboot prompt, pllset command can be used. E.g.:

```
pllset arm 19 1 2
```

Here the calculation example assuming the refclkmain is 125MHz please use the refclkmain value mentioned above when changing on EVM):

```
125000000*19/(1*2) = 1187500000 Hz
```

At Uboot prompt, getclk command can be used to verify the Tetris frequency setting. The argument for Tetris component is based on the enum clk_e definition in clock_k2x.h. For K2H, the Tetris speed can be read by the following command

```
>getclk 2
```

For K2E, the Tetris speed is the same as core PLL and can not be changed. The core_pll_clk defined in enum clk_e of clock_k2e.h is 0. So, for K2E, the Tetris speed can be read by the following command

```
>getclk 0
```

3. To change the Core PLL clock, you need to update the core_pll_config[] in board/ti/ks2_evm/board_k2x.c by replacing CORE_PLL_1228 with appropriate clock define.

For example, if current core speed of K2H is at 800MHz, modify the core_pll_config setting in board_k2hk.c

```
from  { CORE_PLL,     13,     1,      2 }, /* 799 */
to    { CORE_PLL,     625,     32,      2 }, /* 1200 */
```

How to boot Linux using uinitrd?

To boot Linux using a uImage formatted initrd, follow the procedure given here.

1. First generate the uinitrd.bin image using the following command on a Linux Host:

```
mkimage  -A arm -O linux -T ramdisk -C gzip -a 0 -e 0 -n initramfs -d arago-console-image.cpio.gz uinitrd.bin
```

Where arago-console-image.cpio.gz is available as part of the release. Copy uinitrd.bin to the tftp server root directory

2. Make sure the u-boot image is built using latest u-boot-keystone source at git.ti.com. Following commits are required at the minimum:-

```
keystone: dt fix up for uinitrd
keystone2: automate boot up using uinitrd fs
```

3. Upgrade the u-boot image to the EVM.

4. Do the following command after u-boot boots up

```
env default -f -a
setenv serverip <tftp server IP>
setenv tftp_root <Tftp server root directory>
setenv boot uinitrd
saveenv
```

4a. If using LPAE on Linux version below 3.13, and have the patch "keystone2: add env option to do unitrd dt fixup", set the following flag:

```
setenv uinitrd_fixup 1
saveenv
```

5. Issue boot command to boot to Linux. A sample initial part of u-boot part of the bootlog is shown below.

```
## Booting kernel from Legacy Image at 88000000 ...
   Image Name:   Linux-3.8.4-rt2-01184-gf78749b
   Created:      2013-07-05  19:56:56 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3648776 Bytes = 3.5 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 82000000 ...
   Image Name:   initramfs
   Created:      2013-07-01  18:38:53 UTC
   Image Type:   ARM Linux RAMDisk Image (gzip compressed)
   Data Size:    7652966 Bytes = 7.3 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
   Booting using the fdt blob at 0x87000000
   Loading Kernel Image ... OK
OK
   Using Device Tree in place at 87000000, end 8700dae3
```

How to use fixed IP address instead of DHCP?

1. U-boot IP address

```
 To statically configure an IP address for u-boot, use the setenv command:
  setenv ipaddr <ip_address>
```

2. If static IP address is configured in u-boot, then it is not applicable to use dhcp to download files. It is necessary to change all downloading commands in printenv from 'dhcp <file_name>' to 'tftp <file_name>'. and the command examples are as following:

```
 setenv get_fdt_net 'tftp ${addr_fdt} ${tftp_root}/${name_fdt}'
 setenv get_fdt_ramfs 'tftp ${addr_fdt} ${tftp_root}/${name_fdt}'
 setenv get_fs_ramfs 'tftp ${addr_fs} ${tftp_root}/${name_fs}'
 setenv get_kern_net 'tftp ${addr_kern} ${tftp_root}/${name_kern}'
 setenv get_kern_ramfs 'tftp ${addr_kern} ${tftp_root}/${name_kern}'
 setenv get_mon_net 'tftp ${addr_mon} ${tftp_root}/${name_mon}'
 setenv get_mon_ramfs 'tftp ${addr_mon} ${tftp_root}/${name_mon}'
 setenv get_uboot_net 'tftp ${addr_uboot} ${tftp_root}/${name_uboot}'
 setenv get_uboot_ramfs 'tftp ${addr_uboot} ${tftp_root}/${name_uboot}'
```

To revert the commands back to default settings if not using static IP address:

```
 setenv get_fdt_net 'dhcp ${addr_fdt} ${tftp_root}/${name_fdt}'
 setenv get_fdt_ramfs 'dhcp ${addr_fdt} ${tftp_root}/${name_fdt}'
 setenv get_fs_ramfs 'dhcp ${addr_fs} ${tftp_root}/${name_fs}'
 setenv get_kern_net 'dhcp ${addr_kern} ${tftp_root}/${name_kern}'
 setenv get_kern_ramfs 'dhcp ${addr_kern} ${tftp_root}/${name_kern}'
 setenv get_mon_net 'dhcp ${addr_mon} ${tftp_root}/${name_mon}'
 setenv get_mon_ramfs 'dhcp ${addr_mon} ${tftp_root}/${name_mon}'
 setenv get_uboot_net 'dhcp ${addr_uboot} ${tftp_root}/${name_uboot}'
 setenv get_uboot_ramfs 'dhcp ${addr_uboot} ${tftp_root}/${name_uboot}'
```

3. Linux Kernel IP address

```
 To statically configure IP address for Linux Kernel, The args_all  need to be updated/modified for all boot modes:ubi, net, ramfs etc. with the following commands using boot=net
as the example and assuming the static IP address is the same as u-boot environment variable, ipaddr:
```

```
1) Create uboot environment variable for netmask.
   # setenv netmask 255.255.255.0
2) Check environment variable for ipaddr is set.  Set if necessary.
   # printenv ipaddr
3) Check environment variable for gatewayip is set. Set if necessary.
   # printenv gatewayip
4) Modify args_all
   # setenv args_all 'setenv bootargs console=ttyS0,115200n8 rootwait=1 ip=${ipaddr}::${gatewayip}:${netmask}::eth0:off'
5) Remove “ip=dhcp” from args_net
   a)printenv args_net
   b)copy everything except ip=dhcp
   c)setenv args_net ‘<copy args_net from step above without ip=dhcp>’ (quotes are important).
     The command should look like:
     #setenv args_net 'setenv bootargs ${bootargs} rootfstype=nfs root=/dev/nfs rw nfsroot=${serverip}:${nfs_root},${nfs_options}'
6) Save the uboot environment
   #saveenv
7) Boot the EVM
   #boot
```

If using multiple interfaces, DHCP and NFS filesystem

To make sure the interface your NFS server is connected to is the first interface to boot in the bootargs explained in the section above, you may assign "ip=:::::eth0:dhcp". This will ensure eth0 is used for NFS mount.

How to boot from USB Flash drive?

On Ubuntu Host

I - Install gparted

```
$sudo apt-get install gparted
```

Used GParted 0.11.0 for this test. So exact steps may vary if version is different

II - Partition the device

Use partition 1 for boot images and Partition 2 for rootfs

insert the usb flash drive to usb slot

```
$sudo gparted
```

The GUI will show up now. Choose correct device (/dev/sdx) at the top right corner (shows the correct size of the usb drive). This example uses /dev/sdc1 for partion 1 and /dev/sdc2 for partition 2

Warning!! Before you proceed, make sure you selected the correct device, otherwise it may format the active harddisk partition of your PC

Got to top level "Partition" pulldown menu and select Unmount to unmount the device. If there is existing partition, delete the same (Partition -> delete)

Now Partition and filesystem status shows as unallocated

1. Create fat32 partion for boot (contains boot images)

Partition -> new

New size = 32MiB
File system = fat32
label = boot
Keep rest of the fields default
click add to add partition
select the partiton #1 just created and format it (Partition -> format to).

select fat32 format

2. Create ext4 format for rootfs (contains root filesystem)

select the unallocate partition
Partition -> new

File system = ext4
label = rootfs
Keep rest of the fields default
click add to add partition
select the partiton #2 just created and format it (Partition -> format to).

select ext4 format

3. Apply the changes and Quit

Edit -> Apply All Operations
GParted -> Quit

III - Copy images and rootfs files to partitions

Unmount if the devices are automounted

```
$sudo umount  /dev/sdc1
$sudo umount  /dev/sdc2
```

1. Copy images to partition #1 (boot)

```
$sudo mount -t vfat /dev/sdc1 /mnt/test
```

Assume all images are available at the current directory

```
$sudo cp skern-keystone-evm.bin /mnt/test/
$sudo cp uImage-k2hk-evm.dtb /mnt/test/
$sudo cp uImage-keystone-evm.bin /mnt/test/
$ls /mnt/test
skern-keystone-evm.bin  uImage-k2hk-evm.dtb  uImage-keystone-evm.bin
$sudo umount /dev/sdc1
```

2. Copy rootfs files to partition #2 (rootfs)

```
$mdir test
$cd test
$gunzip <path of arago-console-image.cpio.gz>
$sudo cpio -i -F ../arago-console-image.cpio
$sudo mount -t ext4 /dev/sdc2 /mnt/test
$cd ..
$sudo cp -r test/* /mnt/test
$ls /mnt/test/
bin  boot  dev  etc  home  init  lib  lost+found  media  mnt  proc  sbin  srv  sys  tmp  usr  var
$sudo umount /dev/sdc2
```

On EVM

Requires Linux Kernel image with EXT4 support

Setup u-boot env variables

Insert USB flash drive to usb slot on EVM and Power ON EVM Type the following commands to setup the env for usb boot

```
>setenv boot usb
>setenv args_usb 'setenv bootargs ${bootargs} rootfstype=ext4 root=/dev/sda2 rw'
>setenv init_usb 'usb start; run set_fs_none args_all args_usb'
>setenv get_fdt_usb 'fatload usb 0:1 ${addr_fdt} ${name_fdt}'
>setenv get_kern_usb 'fatload usb 0:1 ${addr_kern} ${name_kern}'
>setenv get_mon_usb 'fatload usb 0:1 ${addr_mon} ${name_mon}'
>saveenv
>boot
IMPORTANT: Make sure "rootwait" (NOT "rootwait=1") appears in args_all
```

Here is the log using a USB 3.0 device. Boot takes about 1 minute

```
U-Boot 2013.01-00009-gd274e3f-dirty (Aug 29 2013 - 12:10:33)
```

```
I2C:   ready
DRAM:  1 GiB
NAND:  512 MiB
Net:   TCI6638_EMAC, TCI6638_EMAC1
Hit any key to stop autoboot:  0
(Re)start USB...
USB0:   No power optimization available
Register 2001040 NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus 0 for devices...
2 USB Device(s) found
      scanning usb for storage devices... 1 Storage Device(s) found
reading uImage-k2hk-evm.dtb
44562 bytes read in 1067 ms (40 KiB/s)
reading skern-keystone-evm.bin
45056 bytes read in 949 ms (45.9 KiB/s)
reading uImage-keystone-evm.bin
3989320 bytes read in 48025 ms (81.1 KiB/s)
## installed monitor, freq [194560000], status 0
## Booting kernel from Legacy Image at 88000000 ...
   Image Name:   Linux-3.8.4-rt2-01225-g62655f8
   Created:      2013-08-30  14:33:02 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3989256 Bytes = 3.8 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
   Booting using the fdt blob at 0x87000000
   Loading Kernel Image ... OK
OK
   Using Device Tree in place at 87000000, end 8700de11
```

```
Starting kernel ...
```

```
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 3.8.4-rt2-01225-g62655f8 (a0868495@ares-ubuntu.am.dhc
p.ti.com) (gcc version 4.7.3 20130226 (prerelease) (crosstool-NG linaro-1.13.1-4.7
-2013.03-20130313 - Linaro GCC 2013.03) ) #3 SMP PREEMPT RT Fri Aug 30 10:32:50 ED
T 2013
[    0.000000] CPU: ARMv7 Processor [412fc0f4] revision 4 (ARMv7), cr=30c7387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] Machine: KeyStone2, model: Texas Instruments Keystone 2 SoC
[    0.000000] switching to high address space at 0x800000000
```

```
[    0.000000] cma: CMA: reserved 16 MiB at 1f000000
[    0.000000] Memory policy: ECC disabled, Data cache writealloc
[    0.000000] PERCPU: Embedded 9 pages/cpu @c0bd2000 s12864 r8192 d15808 u36864
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages
: 130048
[    0.000000] Kernel command line: console=ttyS0,115200n8 rootwait=1 rootwait=1 r
ootfstype=ext4 rootwait root=/dev/sda2 rw ip=dhcp
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] __ex_table already sorted, skipping sort
[    0.000000] Memory: 512MB = 512MB total
[    0.000000] Memory: 495164k/495164k available, 29124k reserved, 0K highmem
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff000000   ( 488 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc070243c   (7146 kB)
[    0.000000]       .init : 0xc0703000 - 0xc0742240   ( 253 kB)
[    0.000000]       .data : 0xc0744000 - 0xc078b8e0   ( 287 kB)
[    0.000000]        .bss : 0xc078b8e0 - 0xc07b63ac   ( 171 kB)
[    0.000000] SLUB: Genslabs=11, HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nod
es=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] ipc irq: irqchip registered, range 512-539
[    0.000000] main_pll_clk rate is 1167360000, postdiv = 2, mult = 18,prediv = 0
[    0.000000] pll_clk parent_rate(122880000 Hz), rate(327680000 Hz),postdiv = 6,
mult = 15, prediv = 0
[    0.000000] tci6614-timer: no matching node
[    0.000000] Architected local timer running at 194.56MHz (phys).
[    0.000000] Switching to timer-based delay loop
[    0.000000] sched_clock: 32 bits at 194MHz, resolution 5ns, wraps every 22075ms
[    0.000000] Console: colour dummy device 80x30
[    0.000145] Calibrating delay loop (skipped), value calculated using timer freq
uency.. 389.12 BogoMIPS (lpj=1945600)
[    0.000148] pid_max: default: 4096 minimum: 301
[    0.000300] Mount-cache hash table entries: 512
[    0.011993] CPU: Testing write buffer coherency: ok
[    0.012169] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.012204] Setting up static identity map for 0x80517268 - 0x8051729c
[    0.182030] CPU1: Booted secondary processor
[    0.182044] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.261490] CPU2: Booted secondary processor
[    0.261504] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.341575] CPU3: Booted secondary processor
[    0.341590] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.341652] Brought up 4 CPUs
[    0.341674] SMP: Total of 4 processors activated (1556.48 BogoMIPS).
[    0.360841] NET: Registered protocol family 16
[    0.384047] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.395928] keystone2_pcie_serdes_setup
[    0.397997] keystone2_pcie_serdes_setup done
[    0.398015] hw-breakpoint: found 5 (+1 reserved) breakpoint and 4 watchpoint re
gisters.
[    0.398021] hw-breakpoint: maximum watchpoint size is 8 bytes.
[    0.411553] bio: create slab <bio-0> at 0
[    0.411792] keystone-pcie: keystone_pcie_rc_init - start
[    0.411941] MEM 0x0000000050000000..0x00000005fffffff -> 0x0000000050000000
[    0.411952] IO 0x0000000024000000..0x0000000024003fff -> 0x0000000000000000
[    0.411993] pcie - number of legacy irqs = 4
[    0.412054] pcie - number of MSI host irqs = 8, msi_irqs = 32
[    0.522253] keystone-pcie: Doing PCI Setup...Done
[    0.522260] keystone-pcie: Starting PCI scan...
[    0.522425] PCI host bridge to bus 0000:00
[    0.522438] pci_bus 0000:00: root bus resource [mem 0x50000000-0x5fffffff]
[    0.522446] pci_bus 0000:00: root bus resource [io  0x0000-0x3fff]
[    0.522454] pci_bus 0000:00: No busn resource found for root bus, will use [bus
 00-ff]
[    0.522497] PCI: bus0: Fast back to back transfers enabled
[    0.522515] keystone-pcie: Ending PCI scan...
[    0.522524] keystone-pcie: keystone_pcie_rc_init - end
[    0.522757] vgaarb: loaded
[    0.523196] SCSI subsystem initialized
[    0.523743] usbcore: registered new interface driver usbfs
[    0.523871] usbcore: registered new interface driver hub
[    0.524010] usbcore: registered new device driver usb
[    0.525636] pps_core: LinuxPPS API ver. 1 registered
[    0.525643] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giomett
i <giometti@linux.it>
[    0.525729] PTP clock support registered
[    0.525906] keystone-hwqueue hwqueue.4: qmgr start queue 0, number of queues 81
92
[    0.526024] keystone-hwqueue hwqueue.4: added qmgr start queue 0, num of queues
8192, reg_peek e0840000, reg_status e0804000, reg_config e0806000, reg_region e08
08000, reg_push e0880000, reg_pop e08c0000
[    0.526034] keystone-hwqueue hwqueue.4: qmgr start queue 8192, number of queues
8192
[    0.526144] keystone-hwqueue hwqueue.4: added qmgr start queue 8192, num of que
ues 8192, reg_peek e0900000, reg_status e080a400, reg_config e080c000, reg_region
e080e000, reg_push e0940000, reg_pop e0980000
[    0.527132] keystone-hwqueue hwqueue.4: qos: sched port @8096, drop sched @8000
[    0.528509] keystone-hwqueue hwqueue.4: qos: sched port @6496, drop sched @6400
[    0.529871] keystone-hwqueue hwqueue.4: added pool pool-net: 2048 descriptors o
f size 128
[    0.529882] keystone-hwqueue hwqueue.4: added pool pool-rio: 128 descriptors of
size 256
[    0.529892] keystone-hwqueue hwqueue.4: added pool pool-udma: 1636 descriptors
of size 256
[    0.529902] keystone-hwqueue hwqueue.4: added pool pool-xge: 2048 descriptors o
f size 128
[    0.529912] keystone-hwqueue hwqueue.4: added pool pool-crypto: 512 descriptors
 of size 128
[    0.533290] keystone-hwqueue hwqueue.4: registered queues 0-16383
[    0.533672] keystone-hwqueue hwqueue.4: qos version 0x2000105, magic valid
[    0.534173] keystone-hwqueue hwqueue.4: qos version 0x2000105, magic valid
[    0.541329] keystone-pktdma 2004000.pktdma: registered 24 logical channels, flo
ws 32, tx chans: 9, rx chans: 24
[    0.545326] keystone-pktdma 2a08000.pktdma: registered 24 logical channels, flo
ws 32, tx chans: 32, rx chans: 32, loopback
[    0.546110] keystone-pktdma 2fa1000.pktdma: registered 4 logical channels, flow
s 32, tx chans: 16, rx chans: 16
[    0.546301] Switching to clocksource arch_sys_counter
[    0.577408] NET: Registered protocol family 2
[    0.577857] TCP established hash table entries: 4096 (order: 3, 32768 bytes)
[    0.577936] TCP bind hash table entries: 4096 (order: 4, 114688 bytes)
[    0.578066] TCP: Hash tables configured (established 4096 bind 4096)
```

```
[    0.578105] TCP: reno registered
[    0.578117] UDP hash table entries: 256 (order: 2, 16384 bytes)
[    0.578143] UDP-Lite hash table entries: 256 (order: 2, 16384 bytes)
[    0.578361] NET: Registered protocol family 1
[    0.578574] RPC: Registered named UNIX socket transport module.
[    0.578581] RPC: Registered udp transport module.
[    0.578587] RPC: Registered tcp transport module.
[    0.578592] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.579044] hw perfevents: enabled with ARMv7 Cortex-A15 PMU driver, 7 counters
available
[    0.667192] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[    0.667560] NTFS driver 2.1.30 [Flags: R/O].
[    0.667944] jffs2: version 2.2. (NAND) Â© 2001-2006 Red Hat, Inc.
[    0.668478] msgmni has been set to 999
[    0.669499] NET: Registered protocol family 38
[    0.669789] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 252
)
[    0.669798] io scheduler noop registered
[    0.669804] io scheduler deadline registered
[    0.670000] io scheduler cfq registered (default)
[    0.671576] keystone-udma udma0.5: registered udma device udma0
[    0.734545] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    0.736476] 2530c00.serial: ttyS0 at MMIO 0x2530c00 (irq = 309) is a 16550A
[    1.523400] console [ttyS0] enabled
[    1.527763] 2531000.serial: ttyS1 at MMIO 0x2531000 (irq = 312) is a 16550A
[    1.538861] loop: module loaded
[    1.542184] at24 0-0050: 131072 byte 24c1024 EEPROM, writable, 1 bytes/write
[    1.550398] Generic platform RAM MTD, (c) 2004 Simtec Electronics
[    1.564415] ONFI param page 0 valid
[    1.567913] ONFI flash detected
[    1.571064] NAND device: Manufacturer ID: 0x2c, Chip ID: 0xac (Micron MT29F4G08
ABBDAHC), 512MiB, page size: 2048, OOB size: 64
[    1.582770] Bad block table found at page 262080, version 0x02
[    1.589032] Bad block table found at page 262016, version 0x02
[    1.595089] nand_read_bbt: bad block at 0x000001a20000
[    1.600266] nand_read_bbt: bad block at 0x00001ce60000
[    1.605484] 3 ofpart partitions found on MTD device 30000000.nand
[    1.611596] Creating 3 MTD partitions on "30000000.nand":
[    1.617014] 0x000000000000-0x000000100000 : "u-boot"
[    1.622827] 0x000000100000-0x000000180000 : "params"
[    1.628599] 0x000000180000-0x000008000000 : "ubifs"
[    1.634328] davinci_nand 30000000.nand: controller rev. 2.5
[    1.640645] spi_davinci 21000400.spi: master is unqueued, this is deprecated
[    1.648106] m25p80 spi32766.0: found n25q128a11, expected n25q128
[    1.654221] m25p80 spi32766.0: n25q128a11 (16384 Kbytes)
[    1.659558] 2 ofpart partitions found on MTD device spi32766.0
[    1.665408] Creating 2 MTD partitions on "spi32766.0":
[    1.670562] 0x000000000000-0x000000080000 : "u-boot-spl"
[    1.676874] 0x000000080000-0x000001000000 : "test"
[    1.682874] spi_davinci 21000400.spi: Controller at 0xe0878400
[    1.689789] tun: Universal TUN/TAP device driver, 1.6
[    1.694854] tun: (C) 1999-2004 Max Krasnyansky <maxk@qualcomm.com>
[    1.701654] keystone-netcp 2f00000.netcp: No streaming regs defined
[    1.708067] keystone-netcp 2090000.netcp: cpts rftclk freq not defined
[    1.715696] keystone-netcp 2090000.netcp: Created interface "eth0"
[    1.721904] keystone-netcp 2090000.netcp: dma_chan_name nettx0
[    1.728761] keystone-netcp 2090000.netcp: Created interface "eth1"
[    1.734967] keystone-netcp 2090000.netcp: dma_chan_name nettx1
[    1.742144] keystone-dwc3 2690000.dwc: usbss revision 47914300
[    1.748031] keystone-dwc3 2690000.dwc: mapped irq 425 to virq 608
[    1.956808] xhci-hcd xhci-hcd: xHCI Host Controller
[    1.961718] xhci-hcd xhci-hcd: new USB bus registered, assigned bus number 1
[    1.969739] xhci-hcd xhci-hcd: irq 608, io mem 0x02690000
[    1.975240] usb usb1: New USB device found, idVendor=1d6b, idProduct=0002
[    1.982052] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    1.989300] usb usb1: Product: xHCI Host Controller
[    1.994191] usb usb1: Manufacturer: Linux 3.8.4-rt2-01225-g62655f8 xhci-hcd
[    2.001176] usb usb1: SerialNumber: xhci-hcd
[    2.005922] hub 1-0:1.0: USB hub found
[    2.009692] hub 1-0:1.0: 1 port detected
[    2.013864] xhci-hcd xhci-hcd: xHCI Host Controller
[    2.018765] xhci-hcd xhci-hcd: new USB bus registered, assigned bus number 2
[    2.025955] usb usb2: New USB device found, idVendor=1d6b, idProduct=0003
[    2.032766] usb usb2: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    2.040012] usb usb2: Product: xHCI Host Controller
[    2.044904] usb usb2: Manufacturer: Linux 3.8.4-rt2-01225-g62655f8 xhci-hcd
[    2.051888] usb usb2: SerialNumber: xhci-hcd
[    2.056645] hub 2-0:1.0: USB hub found
[    2.060414] hub 2-0:1.0: 1 port detected
[    2.064649] Initializing USB Mass Storage driver...
[    2.069687] usbcore: registered new interface driver usb-storage
[    2.075711] USB Mass Storage support registered.
[    2.080647] mousedev: PS/2 mouse device common for all mice
[    2.086548] i2c /dev entries driver
[    2.090806] watchdog 22f0080.wdt: heartbeat 60 sec
[    2.097888] keystone-crypto 20c0000.crypto: crypto accelerator enabled
[    2.104879] usbcore: registered new interface driver usbhid
[    2.110470] usbhid: USB HID core driver
[    2.114750]  remoteproc0: 2620040.dsp0 is available
[    2.119643]  remoteproc0: Note: remoteproc is still under development and consi
dered experimental.
[    2.128633]  remoteproc0: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.139060]  remoteproc0: no firmware found
[    2.143397] rproc-user 2620040.dsp0: registered misc device dsp0
[    2.149740]  remoteproc1: 2620044.dsp1 is available
[    2.154631]  remoteproc1: Note: remoteproc is still under development and consi
dered experimental.
[    2.163623]  remoteproc1: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.174048]  remoteproc1: no firmware found
[    2.178379] rproc-user 2620044.dsp1: registered misc device dsp1
[    2.184719]  remoteproc2: 2620048.dsp2 is available
[    2.189612]  remoteproc2: Note: remoteproc is still under development and consi
dered experimental.
[    2.198600]  remoteproc2: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.209064]  remoteproc2: no firmware found
[    2.213487] rproc-user 2620048.dsp2: registered misc device dsp2
[    2.219843]  remoteproc3: 262004c.dsp3 is available
[    2.224734]  remoteproc3: Note: remoteproc is still under development and consi
dered experimental.
[    2.233723]  remoteproc3: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.244149]  remoteproc3: no firmware found
[    2.248479] rproc-user 262004c.dsp3: registered misc device dsp3
[    2.254819]  remoteproc4: 2620050.dsp4 is available
[    2.259712]  remoteproc4: Note: remoteproc is still under development and consi
dered experimental.
[    2.266440] hub 1-0:1.0: unable to enumerate USB device on port 1
```

```
[    2.274811]  remoteproc4: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.285242]  remoteproc4: no firmware found
[    2.289572] rproc-user 2620050.dsp4: registered misc device dsp4
[    2.295905]  remoteproc5: 2620054.dsp5 is available
[    2.300797]  remoteproc5: Note: remoteproc is still under development and consi
dered experimental.
[    2.309787]  remoteproc5: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.320212]  remoteproc5: no firmware found
[    2.324547] rproc-user 2620054.dsp5: registered misc device dsp5
[    2.330885]  remoteproc6: 2620058.dsp6 is available
[    2.335776]  remoteproc6: Note: remoteproc is still under development and consi
dered experimental.
[    2.344765]  remoteproc6: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.355192]  remoteproc6: no firmware found
[    2.359519] rproc-user 2620058.dsp6: registered misc device dsp6
[    2.365870]  remoteproc7: 262005c.dsp7 is available
[    2.370762]  remoteproc7: Note: remoteproc is still under development and consi
dered experimental.
[    2.379750]  remoteproc7: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.390175]  remoteproc7: no firmware found
[    2.394505] rproc-user 262005c.dsp7: registered misc device dsp7
[    2.400557] rproc-user dspmem.3: kick gpio
[    2.404798] rproc-user dspmem.3: registered misc device dspmem
[    2.410817] GACT probability on
[    2.413967] Mirror/redirect action on
[    2.417641] Simple TC action Loaded
[    2.421669] netem: version 1.3
[    2.424734] u32 classifier
[    2.427448]     Performance counters on
[    2.431290]     input device check on
[    2.434958]     Actions configured
[    2.438379] Netfilter messages via NETLINK v0.30.
[    2.443107] nf_conntrack version 0.5.0 (7992 buckets, 31968 max)
[    2.449469] ctnetlink v0.93: registering with nfnetlink.
[    2.455114] IPv4 over IPv4 tunneling driver
[    2.459827] gre: GRE over IPv4 demultiplexor driver
[    2.464719] ip_gre: GRE over IPv4 tunneling driver
[    2.470185] ip_tables: (C) 2000-2006 Netfilter Core Team
[    2.475612] ipt_CLUSTERIP: ClusterIP Version 0.8 loaded successfully
[    2.482023] arp_tables: (C) 2002 David S. Miller
[    2.486697] TCP: cubic registered
[    2.490018] Initializing XFRM netlink socket
[    2.495227] NET: Registered protocol family 10
[    2.500552] NET: Registered protocol family 17
[    2.505028] NET: Registered protocol family 15
[    2.509539] Bridge firewalling registered
[    2.513564] Ebtables v2.0 registered
[    2.517249] 8021q: 802.1Q VLAN Support v1.8
[    2.521741] sctp: Hash tables configured (established 4096 bind 4681)
[    2.528729] NET: Registered protocol family 40
[    2.533347] VFP support v0.3: implementor 41 architecture 4 part 30 variant f r
ev 0
[    2.541048] Registering SWP/SWPB emulation handler
[    2.546470] usb 2-1: new SuperSpeed USB device number 2 using xhci-hcd
[    2.547633] input: gpio_keys.7 as /devices/soc.0/gpio_keys.7/input/input0
[    2.554138] keystone-netcp 2090000.netcp: initializing cpsw version 1.3 (1) SGM
II identification value 0x4ed1
[    2.554301] keystone-netcp 2090000.netcp: Created a cpsw ale engine
[    2.554307] keystone-netcp 2090000.netcp: initialized cpsw ale revision 1.3
[    2.557389] pps pps0: new PPS source ptp0
[    2.557412] cpts rftclk rate(583680000 HZ),mult(1839607018),shift(30)
[    2.597538] keystone-netcp 2090000.netcp: Using Packet Accelerator Firmware ver
sion 0x01030008
[    2.606188] keystone-netcp 2090000.netcp: pa_clk_rate(163840000 HZ),mult(25000)
,shift(12)
[    2.606609] usb 2-1: Parent hub missing LPM exit latency info.  Power managemen
t will be impacted.
[    2.606962] usb 2-1: New USB device found, idVendor=05dc, idProduct=a203
[    2.606966] usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[    2.606970] usb 2-1: Product: JumpDrive
[    2.606973] usb 2-1: Manufacturer: Lexar
[    2.606977] usb 2-1: SerialNumber: 09020000003802162764
[    2.636458] scsi0 : usb-storage 2-1:1.0
[    2.658553] keystone-netcp 2090000.netcp: netcp device eth0 opened
[    2.667093] 8021q: adding VLAN 0 to HW filter on device eth0
[    2.672772] keystone-netcp 2090000.netcp: adding rx vlan id: 0
[    2.679143] keystone-netcp 2090000.netcp: initializing cpsw version 1.3 (1) SGM
II identification value 0x4ed1
[    2.695045] keystone-netcp 2090000.netcp: netcp device eth1 opened
[    2.701867] IPv6: ADDRCONF(NETDEV_UP): eth1: link is not ready
[    2.707719] 8021q: adding VLAN 0 to HW filter on device eth1
[    2.713399] keystone-netcp 2090000.netcp: adding rx vlan id: 0
[    2.736385] Sending DHCP requests ., OK
[    2.786410] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
[    2.806392] IP-Config: Got DHCP answer from 158.218.104.3, my address is 158.21
8.104.111
[    3.636778] scsi 0:0:0:0: Direct-Access     Lexar    JumpDrive        1.00 PQ:
0 ANSI: 5
[    3.645783] sd 0:0:0:0: [sda] 31293440 512-byte logical blocks: (16.0 GB/14.9 G
iB)
[    3.645884] sd 0:0:0:0: Attached scsi generic sg0 type 0
[    3.658873] sd 0:0:0:0: [sda] Write Protect is off
[    3.663853] sd 0:0:0:0: [sda] Write cache: disabled, read cache: disabled, does
n't support DPO or FUA
[    3.674832]  sda: sda1 sda2
[    3.678982] sd 0:0:0:0: [sda] Attached SCSI removable disk
[    4.256594] keystone-netcp 2090000.netcp: removing rx vlan id: 0
[    4.263472] IP-Config: Complete:
[    4.266713]      device=eth0, hwaddr=00:17:ea:f4:46:0b, ipaddr=158.218.104.111,
mask=255.255.255.0, gw=158.218.104.1
[    4.277271]      host=158.218.104.111, domain=am.dhcp.ti.com, nis-domain=(none)
[    4.284603]      bootserver=0.0.0.0, rootserver=0.0.0.0, rootpath=
[    4.290629]      nameserver0=158.218.108.21, nameserver1=157.170.32.67
[    4.305325] EXT4-fs (sda2): mounted filesystem with ordered data mode. Opts: (n
ull)
[    4.313039] VFS: Mounted root (ext4 filesystem) on device 8:2.
[    4.319097] Freeing init memory: 252K
INIT: version 2.88 booting
Starting udev
e or directory
Starting Bootlog daemon: bootlogd.
[    5.869741] EXT4-fs (sda2): re-mounted. Opts: data=ordered
Configuring network interfaces... udhcpc (v1.20.2) started
Sending discover...
Sending select for 158.218.104.111...
Lease of 158.218.104.111 obtained, lease time 3600
```

```
/etc/udhcpc.d/50default: Adding DNS 158.218.108.21
/etc/udhcpc.d/50default: Adding DNS 157.170.32.67
done.
Wed May 15 06:38:00 UTC 2013
Starting telnet daemon.
Starting syslogd/klogd: done
Starting thttpd.
Stopping Bootlog daemon: bootlogd.
```

```
 _____                  _____           _       _
|  _  |___ ___ ___ ___  |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . | |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___| |__|  |_| |___|_| |___|___|_|
              |___|                     |___|
```

```
Arago Project http://arago-project.org keystone-evm ttyS0
```

```
Arago 2013.04 keystone-evm ttyS0
```

```
keystone-evm login: root
```

Why can't I connect to DSP cores from CCS in latest u-boot?

In the latest u-boot, we have DSP cores powered OFF by default. Need to set debug_options=1 U-Boot env variable and reset the board to get this working

```
>setenv debug_options 1
>saveenv
>reset
```

How to turn OFF ARM core 0?

U-Boot support turning OFF core 0. By default Core 1-3 are OFF. The following command can be used to turn OFF core 0

```
>setenv boot net
>run get_mon_net
>run run_mon
>killme
```

How to debug kernel through CCS?

Boot up EVM to u-boot. Startup CCS and Launch Target configuration. connect to ARM core using "connect to target from right click menu" or cntrl-alt-c. To do source level debugging, first enable source look up (right click on *.ccxml on the CCS window and choose "Edit source lookup" option. In the window, Right click on ARM core and choose "add" option. Choose "Path Mapping" in the next window. Assuming w:/linux-keystone on the windows machine is mapped to linux /home/<user-id>/linux-keystone, following entries are required

```
Compilation path \home\<user-id>\linux-keystone
Local file system path w:\linux-keystone
```

Click ok and complete the source lookup settings.

Boot to Linux. As soon as the Linux start booting, click suspend button on CCS window.

Now Load the symbol file, vmlinux that is available in the root directory of the kernel tree where the kernel build is done. Choose "Run" from the top level menu and then "load" -> "Load Symbol" and choose vmlinux.

Once loading completes, the source code will be shown with the PC pointing to current location of code execution. May put break point to stop kernel at specific point in Linux kernel boot up sequence.

How to workaround the UBI boot "bad image sequence number" problem ?

Sometimes the UBI image is not properly rebooted, the last PEB may get corrupted and cause the "bad image sequence number" error. This mostly happen when users logically configure a ubifs partition with the size smaller than the available NAND space size. E.g. the actual NAND size is 2GB on K2L EVM, but users only use 512MB for bootloader/env params/ubifs partitions.

When the problem happens, there is a workaround to scrub the entire NAND device by using the following U-boot command:

```
nand scrub.chip
```

How to boot a custom images on Keystone2 devices?

Examples for creation and flashing custom boot images on devices in Keystone2 family are described in the wiki article KeystoneII_Boot_Examples (http://processors.wiki.ti.com/index.php/KeystoneII_Boot_Examples)

The article provides single stage and multi-stage boot examples from different boot modes. It also demonstrates formating uboot images for different boot media using the boot utilities provided in the software package.

Trouble shooting and FAQ

For K2e EVM, not able to do ">mon_install 0x0c5f0000" and stuck there ?

Probably, the MSMC RAM address might be not right. The MSMC RAM address for loading the *.dtb, uImage, skern is given below.

Please do "> print " to display the image loadable addresses and all the values of the env variables before loading the images to the EVM.

For K2E:-

>tftpboot 0x87000000 k2e-evm.dtb

>tftpboot 0x88000000 uImage-keystone-evm.bin

>tftpboot 0x0c140000 skern-k2e.bin

>mon_install 0x0c140000

>bootm 0x88000000 - 0x87000000

For K2H:-

>tftpboot 0x87000000 uImage-k2hk-evm.dtb

>tftpboot 0x88000000 uImage-keystone-evm.bin

>tftpboot 0x0c5f0000 skern-k2hk.bin

>mon_install 0x0c5f0000

>bootm 0x88000000 - 0x87000000

In CCS version: 5.5.0.00077, how to do target configuration in *.ccxml file as I do not see K2E option under "Board or device" column?

User has to install the emulation packages, "ti_emupack_keystone2_setup_1.1.0.0.bin" for linux or "ti_emupack_keystone2_setup_1.1.0.0.exe" for Windows; After which it will list the "66AK2E05" in Board or device column.

you can download those packages from this link:http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/index_FDS.html

After loading the loadlin-evm-uboot.js file, not able to connect to target.

In loadlin-evm-uboot.js file, please check that the session name matches correctly.If not edit it.

For example, change like below:

//var sessionName = "Texas Instruments XDS2xx USB Emulator_0/CortexA15_1";

var sessionName = "Texas Instruments XDS2xx USB Emulator_0/arm_A15_0";

MCSDK 3.01.04.07 YOCTO EDMA3LLD build error

Error Log:

| chmod a+x maketemp_configuro_cmd_c6xdsp.bat | ./maketemp_configuro_cmd_c6xdsp.bat | ./maketemp_configuro_cmd_c6xdsp.bat: 1: ./maketemp_configuro_cmd_c6xdsp.bat: -e: not found | make[2]: *** [xdc_configuro] Error 127

Fix:

Use the following command to fix the problem.

host$ echo "no" | sudo dpkg-reconfigure -f teletype dash

{{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum	Keystone= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article* ***MCSDK UG Chapter Exploring*** *here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article* ***MCSDK UG***	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***MCSDK UG Chapter Exploring*** *here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***MCSDK UG Chapter***	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the*	*For techni please po questions http://e2e. Please po comments article* ***MC Chapter E*** *here.* }}

Please post only comments related to the article **MCSDK UG Chapter Exploring** here.

Please post only comments related to the article **MCSDK UG Chapter Exploring** here.

MCSDK UG Chapter Exploring *here.*

Chapter Exploring *here.*

Exploring *here.*

article ***MCSDK UG Chapter Exploring*** *here.*

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity